UNIVERSITÉ DE MONTRÉAL

ÉVALUATION DE PERFORMANCE DES MALICIELS

PIERRE-MARC BUREAU
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION DU DIPLÔME DE
MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
DÉCEMBRE 2006

Your file Votre référence
ISBN: 978-0-494-25535-3
Our file Notre référence
ISBN: 978-0-494-25535-3

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

ÉVALUATION DE PERFORMANCE DES MALICIELS

présenté par : BUREAU Pierre-Marc
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury constitué de :

M. DAGENAIS Michel, Ph.D., président
M. FERNANDEZ José M., Ph.D., membre et directeur de recherche
M. BOYER François-Raymond, Ph.D., membre

À l'Éducation et au Savoir, points de départ pour l'Autonomie et la Liberté.

Remerciements

Je tiens à remercier mon directeur de recherche, M. José M. Fernandez pour son soutien logistique et tous les bons conseils qu'il m'a prodigués dans le cadre de mes recherches. C'est au cours des nombreuses discussions qui ont agrémenté les deux dernières années que j'ai beaucoup appris sur le domaine de la sécurité informatique, autant au niveau de la gouvernance qu'au niveau technique.

J'aimerais remercier M. Abderahim Alikacem sans qui je n'aurais probablement pas découvert la puissance des logiciels libres. C'est M. Alikacem qui m'a permis de découvrir la recherche scientifique en informatique. Sa motivation et sa dévotion sont des qualités qu'il est très difficile d'oublier. Finalement, M. Alikacem m'a aidé à travailler en collaboration avec Hexago et à faire la connaissance de M. Marc Blanchet que je remercie pour ses nombreux conseils et son ouverture d'esprit.

Je tiens à remercier M. Andrew Lee qui a su nous aider dans nos recherches en nous prodiguant de nombreux conseils sur l'éthique de la recherche sur les maliciels et la perception de l'industrie des logiciels d'antivirus.

Finalement, je n'aurais pu conduire mes recherches et en profiter comme je l'ai fait sans le soutien de ma famille, de mes amis et de ma copine Marie-Claude.

Résumé

Les logiciels malicieux, aussi appelés maliciels, sont des programmes créés avec des intentions malicieuses. Leur utilisation est en croissance sur l'Internet et leur efficacité suit aussi une trajectoire ascendante. Leur création et leur comportement n'ont rien de magique. Ce sont des programmes créés par des êtres humains et ils sont utilisés pour atteindre des objectifs qui vont de la gloire personnelle au détournement de fonds. Avec l'adoption généralisée de l'informatique, les gains engendrés par l'utilisation de maliciels sont gigantesques. Certains affirment qu'il est présentement plus profitable d'effectuer des fraudes en ligne à l'aide de maliciels que de faire le trafic d'héroïne !

Plusieurs chercheurs et plusieurs compagnies se sont attaqués au problème des logiciels malicieux et ont proposé des solutions très intéressantes pour contrer ces menaces. Par contre, la plupart des techniques de défense qui existent sont mal adaptées pour faire face à l'évolution des maliciels. Les outils défensifs fonctionnent bien contre les menaces connues mais sont souvent incapables de détecter et de contrer les menaces inconnues, souvent appelées *Zero Day*. En plus de ne pas toujours détecter les attaques conduites à l'aide de logiciels malicieux, les systèmes de défense ont souvent un taux de faux positifs élevé, ce qui diminue la confiance des utilisateurs.

Les recherches présentées dans ce document visent à mieux comprendre les menaces posées par les maliciels et à prédire une partie de leur évolution. Pour ce faire, nous posons premièrement une liste d'objectifs qui poussent les programmeurs à créer des maliciels. Ces objectifs vont de la simple recherche de reconnaissance par les pairs à la fraude économique dont nous sommes de plus en plus témoins. En plus des objectifs présentés, nous proposons une liste d'indices qui peuvent être mesurés pour déterminer avec quelle efficacité un maliciel atteint ses objectifs.

Les performances des maliciels ne dépendent pas seulement des indices de performance. Les performances des logiciels malicieux sont aussi influencées par l'environnement dans lequel il évolue et, bien sûr, par les caractéristiques du maliciel. C'est pour exprimer cette relation que nous présentons un modèle à trois dimensions. Nous utilisons aussi le modèle de la boucle OODA pour décrire les caractéristiques des maliciels. Ce modèle classifie les caractéristiques d'une unité selon quatre phases : l'observation, l'orientation, la décision et l'action.

Nous illustrons l'utilisation du modèle OODA et l'étude des relations entre les performances, l'environnement et les caractéristiques d'un maliciel dans un exemple concret d'un maliciel évoluant dans un réseau où un filtrage entre les sous réseaux est utilisé. Nous proposons un modèle mathématique basé sur un processus de Markov pour montrer que le filtrage a une influence sur la vitesse de propagation d'un maliciel.

Afin de valider notre modèle mathématique, nous avons créé un cadre de prototypage de maliciels qui est utilisé pour émuler le comportement d'un maliciel dans un vrai réseau. Nous avons effectué trois expériences dans lesquelles un maliciel tente de se propager dans un réseau composé de trente systèmes informatiques et où l'on peut facilement calculer ses performances. La première expérience a été effectuée dans un réseau où un système interconnecte deux sous-réseaux. La deuxième représente quatre systèmes interconnectant deux sous-réseaux et la troisième expérience a été effectuée dans un réseau où toutes les machines peuvent communiquer entre elles.

Les résultats expérimentaux présentés dans ce document ne sont pas les seules contributions scientifiques produites par nos recherches. L'expression des caractéristiques des maliciels à l'aide de la boucle OODA nous a permis de montrer que les capacités d'orientation et de décision sont très limité. Cette faiblesse nous laisse croire que les auteurs de maliciels risquent d'améliorer ces facettes de leurs logiciels et que nous devons nous préparer à faire face à des maliciels plus "intelligents". Le cadre de prototypage développé est facilement réutilisable pour créer des prototypes de logiciels malicieux qui permettront de mieux comprendre les combinaisons et les choix d'implémentation qui doivent être faits par les programmeurs de maliciels. Ce cadre montre aussi que la modélisation des caractéristiques d'un maliciel à l'aide de la boucle OODA est pertinente puisque des prototypes sont facilement créés. De plus, notre modèle mathématique décrivant la relation entre le filtrage dans un réseau et la vitesse de propagation d'un maliciel permet d'émettre des recommandations lors de la configuration de nouveaux réseaux pour les rendre plus tolérants aux infections. Ces recommandations permettent de trouver un compromis entre la facilité d'utilisation d'un réseau et le temps qui sera nécessaire à un virus pour infecter les systèmes composant ce réseau. Nos recherches montrent qu'il est possible d'avoir un impact sur la performance des maliciels. Il est possible, en configurant bien les systèmes de défense qui sont disponibles, de freiner la propagation d'un maliciel encore inconnu et d'augmenter les chances d'enrayer rapidement l'épidémie.

Abstract

Malicious software, also called malware, is a program category created with malicious intents. These programs are widely used on the Internet and their effectiveness at stealing money from users is on the rise. The creation and behaviour of malware is not magic. These programs are created by humans and are used to reach objectives that range from fame and glory to financial fraud and money laundering. With the general adoption of electronic commerce, it is now possible to acquire huge amounts of money by performing frauds with malware. Some expert say that the potential gains from malware operations is presently greater than potential gains from trafficking heroin!

Research groups and companies have showed interest for the problem of malware and put forward interesting counter-measures. The downside of these counter-measures is that they are not suited to face the evolution of malware. Most of the defensive systems we use today work well against known threats but are often unable to detect unknown danger and protect the users. The trust in detection and protection systems is also impacted negatively by their high rate of false positive alarms.

This document presents research aimed towards the comprehension of threats posed by malware and the prediction of their evolution. To understand the evolution of malware, we propose a list of objectives malicious programmers are trying to reach when creating malicious software. In addition to the objectives, we present a list of performance criteria that can be observed to evaluate the effectiveness with which a malware reaches its objectives.

Malware performance does not only depend on their performance criteria. The performance of malicious software is also influenced by its characteristics and by the environment in which it operates. We present a three factors model we created to express the relation between performance criteria, environment and malware characteristics. Furthermore, we created a second model to express the characteristics of malware based on the OODA loop. The OODA loop model divides malware characteristics in four phases : observation, orientation, decision and action.

We use the OODA loop to study the relation between the performance, environ-

ment and characteristic of a malware with a concrete example. Our example involves a malware epidemic within a network where filtering is enforced between subnets. We put forward a mathematical model based on Markov processes to show that network filtering has influence on the spreading speed of malware.

To validate our mathematical model, we developed a malware emulation framework (MEF). This framework lets researchers create emulation agents that can be used to emulate the behaviour of malware inside a computer network. The emulation agents are self replicating Ruby scripts loaded with monitoring and security features in order to perform secure and precise experiments. We used our framework to generate three scenarios where agents would propagate inside a network composed of thirty systems and where we could easily monitor their speed of propagation. The first scenario involves a network of thirty machines without any network filtering. The second scenario was performed using two separate networks with one system connected to both. Finally, the third scenario was done with two networks and four interconnected systems.

The experimental results showed in this document are only part of the contributions of our researches. The expression of malware characteristics with the OODA loop leads us to the conclusion that certain phases of malware behaviour are not at their best. The phases that lack development are the orientation and decision process. This conclusion leads us to believe that we should prepare for more "intelligent" malware in the near future. The practical experiments we have conducted is valuable to the research community because it proves that the OODA loop model works since our agents are modelled using this concept. Furthermore, the documentation on how to prepare a virtual network and conduct an experiment will be useful for further research.

The mathematical model we have developed is a valuable tool for network administrators seeking advice on finding a compromise between the usability of their network and its defence posture against malware attacks. Our research show that it is possible to decrease the performance of malware by changing the characteristics of the environment in which they evolve.

Table des matières

Liste des tableaux

Liste des figures

Liste des sigles et abréviations

AV	Antivirus
DNS	Domain Name Server
DOS	Denial of Service
DDOS	Distributed Denial of Service
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IIS	Internet Information Services
IM	Instant Messaging
IP	Internet Protocol (version 4)
IPS	Intrusion Prevention System
IPv6	Internet Protocol, version 6
IRC	Internet Relay Chat
HTTP	Hyper Text Transfer Protocol
P2P	Peer-to-peer
SQL	Standard Query Language
VBA	Visual Basic for Applications
VM	Virtual Machine

CHAPITRE 1

Introduction

Depuis leur apparition au milieu du siècle dernier, les systèmes d'information n'ont cessé d'évoluer. Leur utilisation est passée de calculs relativement simples dans le domaine militaire au décodage du génome humain. Les premiers ordinateurs étaient extrêmement volumineux et consommaient des quantités monstrueuses d'électricité. De nos jours, nous transportons ces mêmes ordinateurs dans nos poches pour écouter notre musique ou communiquer avec nos amis.

Avec l'adoption générale de l'informatique comme moyen pour améliorer notre qualité de vie, de toutes nouvelles problématiques se sont posées. En effet, les constructeurs se sont souvent vu imposer un rythme de conception et de production très exigeant. C'est, entre autres, ce rythme de production qui a fait en sorte que certaines fonctionnalités ont été mal implantées ou, du moins, mal vérifiées. Comme l'être humain, les ordinateurs sont donc loin d'être parfaits. Ces systèmes contiennent plusieurs failles qui peuvent être exploitées par des attaquants, humains ou automatisés.

Nous avons pendant longtemps appelé les attaquants automatisés des *virus informatiques*. Ces bouts de programmes qui se multiplient sans que les utilisateurs ne s'en rendent compte ont causé des dommages importants dans les dernières années. Les dernières épidémies ont été foudroyantes : le ver informatique *Slammer* a infecté la presque totalité des victimes potentielles sur l'Internet en moins d'un quart d'heure. Il a causé des centaines de milliers de dollars de pertes économiques en rendant des systèmes non disponibles pendant plusieurs heures. Avec le temps et l'évolution des virus informatiques, ces petits programmes se sont diversifiés et nous devons maintenant les séparer en plusieurs familles pour être précis lors de leur description. Nous avons maintenant affaire à des vers informatiques, des logiciels espions, des portes dérobées et des chevaux de Troie.

Dans ce chapitre, nous exposerons d'abord les définitions et concepts de base nécessaires à la compréhension de cet écrit. Nous présenterons ensuite les éléments de la problématique qui justifient cette étude. Nous décrirons les objectifs de recherche

pour terminer avec un tour d'horizon de l'organisation des chapitres qui suivent.

1.1 Définitions et concepts de base

Nous caractérisons tous les logiciels qui ont été conçus dans un but malicieux comme *maliciel*[1]. Le terme maliciel désigne donc une catégorie de programmes informatiques qui ont été créés pour détruire, endommager ou détourner l'utilisation légitime d'un système informatique. La famille des maliciels englobe plusieurs types de logiciels malicieux comme les virus, les vers, les chevaux de Troie et les portes dérobées.

Les virus informatiques forment la plus vieille famille de maliciel. Ils sont caractérisés par le fait qu'ils se copient eux-même sans l'intervention de l'utilisateur. C'est à dire que ce sont des parties de programmes qui se *reproduisent*. Les virus ne sont pas des programmes informatiques complets, ils sont plutôt des parties de programmes qui résident dans d'autres objets du système informatique. Les virus informatiques survivent uniquement grâce à l'existence des autres objets. Ils sont clairement des parasites des autres types de programmes existant sur les ordinateurs.

Les vers informatiques sont beaucoup plus indépendants que les virus. Ils se propagent par eux-mêmes sur les réseaux informatiques, la plupart du temps en exploitant des failles de sécurité dans les systèmes d'exploitation ou les programmes installés sur les systèmes.

Les chevaux de Troie, pour leur part, sont des programmes que l'utilisateur utilise de façon consciente mais qui effectuent aussi des opérations malveillantes à l'insu de celui-ci. Un exemple de cheval de Troie serait un jeu de cartes que l'on téléchargerait de l'Internet. Ce programme peut être complètement fonctionnel mais, quand l'ordinateur n'est plus utilisé, envoyer, par courriel à un attaquant, des fichiers qu'il trouve sur un disque dur. Contrairement aux virus et aux vers, le cheval de Troie ne se multiplie pas par lui-même, il doit être installé par un utilisateur.

La porte dérobée est, comme le cheval de Troie, un maliciel qui ne se propage pas par lui-même. La porte dérobée est souvent installée par un attaquant après qu'il ait gagné l'accès à un système informatique. La porte dérobée sert à conserver l'accès au système pour l'attaquant même si un administrateur corrige les failles de sécurité qui

[1]Ce terme est une adaptation du terme anglais *malware*, qui provient de l'expression *malicious software*.

étaient présentes lors de l'attaque.

Tous ces types de maliciels ne sont pas apparus par hasard sur le grand Réseau. Tous ces programmes ou parties de logiciels sont créés par des programmeurs pour des raisons qui varient beaucoup. De plus, les modifications que prennent certains maliciels sont souvent effectuées par des programmeurs qui veulent utiliser le travail des autres pour s'enrichir, se faire connaître ou voler de l'information.

Les logiciels malicieux existent depuis plus de vingt-cinq ans. Plusieurs compagnies et groupes de chercheurs ont proposé une multitude de moyens qui peuvent être utilisés pour contrer cette menace. De ces recherches sont nés les logiciels anti-virus (AV) et de détection d'intrusion (IDS). Ces deux types de systèmes reposent principalement sur une base de données de signatures pour détecter les menaces. Les AV et IDS observent les informations qui sont consultées par les utilisateurs et avisent les personnes concernées quand ils détectent des informations qui correspondent à une des signatures qu'ils ont en mémoire. Les dernières évolutions des AV et IDS incluent maintenant l'utilisation d'heuristiques et d'algorithmes d'apprentissage. Par exemple. un IDS peut observer le trafic sur un réseau informatique pendant une certaine période de temps pour comprendre quelles sont les habitudes d'utilisation des utilisateurs. Quand il commence à détecter les menaces, il identifie comme menace tous les comportements qui diffèrent de ceux observés précédemment.

Les systèmes de détections basés sur les signatures ont malheureusement beaucoup de faiblesses. Par définition, ces systèmes peuvent seulement détecter des menaces connues, puisqu'ils fondent leur détection sur les signatures qui se trouvent dans leurs bases de données.

Toute nouvelle menace est donc difficilement détectable à l'aide d'un système basé sur les signatures. De plus, les signatures gardées en mémoire sont facilement contournées ; les attaquants n'ont qu'à effectuer des changements mineurs dans les programmes qu'ils ne veulent pas voir détecter. Par exemple, si un système détecte tous les fichiers contenant le mot "Maliciel", il ne détectera pas le mot "maliciel", même si celui-ci a la même signification. Les techniques de détection par heuristiques et algorithmes d'apprentissages ont permis d'augmenter la fiabilité de détection des menaces mais, dès qu'une variation importante de comportement est observée, les solutions de détection ne la reconnaissent pas et manquent à leur tâche de protection. Finalement, les systèmes de détection présentement en utilisation génèrent beaucoup de faux positifs, c'est à dire qu'ils alertent l'utilisateur qu'un acte malveillant

a lieu tandis qu'en réalité, rien de spécial n'est à signaler. Cette quantité élevée de faux-positifs fait perdre du temps aux utilisateurs mais, pire encore, ces faux-positifs diminuent la confiance que les utilisateurs ont dans les systèmes de détection.

1.2 Objectifs de recherche

Nos recherches visent à améliorer nos connaissances des maliciels et de leur évolution pour élaborer des systèmes de défenses qui réagiront mieux aux nouvelles attaques, auront moins de faux-positifs et seront plus difficiles à contourner.

Pour mieux comprendre les maliciels, nous voulons premièrement proposer une taxonomie de ceux-ci. Cette taxonomie devra être à la fois très flexible pour décrire, et peut-être même déduire, les évolutions futures et à la fois assez précise pour être utilisée de façon efficace. Nous validerons l'efficacité de ce cadre en l'utilisant pour caractériser quelques spécimens de maliciels qui ont marqué l'histoire.

Pour évaluer l'efficacité des mécanismes de défense contre les maliciels, nous devons commencer par définir la performance d'un maliciel. Comment savoir si un mécanisme de défense a fonctionné si nous ne sommes pas capables d'exprimer clairement ce qui rend une souche de maliciel efficace ? Pour étudier la performance des maliciels, nous proposons une liste d'objectifs qui poussent les programmeurs à créer des logiciels malicieux. Nous associons à ces objectifs des indices de performance précis qui seront plus facilement observables.

Le troisième élément qui doit être considéré lors de l'évaluation des performances des maliciels est l'environnement dans lequel ceux-ci évoluent. Nous voulons mieux comprendre les liens entre les performances, les caractéristiques et l'environnement des maliciels.

À titre d'exemple, nous voulons créer un modèle mathématique qui décrit la relation entre le filtrage dans un réseau et la vitesse de propagation des maliciels. Ce modèle qui exprime la relation entre l'environnement et un indice de performance est un exemple du type d'études qui peut être effectué à l'aide des modèles et outils développés dans le cadre de nos recherches. Nous validons ce modèle de vitesse de propagation en effectuant des expériences *in vivo*, c'est à dire dans un environnement contrôlé composé de dizaines d'ordinateurs où évolue un agent qui émule le comportement d'un maliciel. La création et la validation de ce modèle sont utiles pour caractériser la sécurité des réseaux quand vient le temps d'évaluer leur résistance aux

attaques conduites à l'aide de maliciels.

Les résultats de notre étude portent sur l'effet du filtrage réseau sur la propagation des maliciels nous permettra de donner des recommandations lors de la configuration de réseaux. On pourra plus facilement identifier le compromis qui doit être fait entre la défense contre les logiciels malicieux et l'aisance d'utilisation du réseau par ses utilisateurs. Malgré le fait que notre travail porte principalement sur l'étude du comportement des maliciels et de leurs interactions avec leur environnement, nous devons toujours garder à l'esprit que nos recherches visent à développer des procédures et des outils efficaces pour sécuriser les réseaux informatiques. Nous cherchons à créer des outils et des procédures pour optimiser l'utilisation des systèmes de défense actuels. Cette optimisation permettra de diminuer les performances des maliciels. C'est à l'aide de ces procédures et de ces outils que nous mettrons en échec la menace posée par les attaquants du monde informatique et leurs logiciels malicieux.

1.3 Plan du mémoire

Ce mémoire est divisé en six chapitres. Le deuxième chapitre effectue une revue de la littérature portant sur les logiciels malicieux et les techniques qui ont été développées pour les observer et pour s'en défendre. Le troisième chapitre expose le modèle structurant sur la performance des maliciels que nous proposons. C'est dans ce chapitre que nous abordons les caractéristiques des maliciels (en utilisant la boucle OODA), leurs performances ainsi que l'environnement dans lequel ils évoluent. Le chapitre suivant montre une application des modèles théoriques à un exemple concret. Ce chapitre étudie la relation entre un indice de performance (la vitesse de propagation) et une caractéristique de l'environnement réseau (le filtrage). Le chapitre quatre montre un modèle théorique développé pour exprimer cette relation et les résultats obtenus à l'aide de ce modèle. Le chapitre 5 expose la description et les résultats que nous avons obtenus en effectuant les expérimentations visant à valider notre modèle théorique sur le filtrage entre sous-réseaux. Pour terminer, le chapitre 6 présente les conclusions tirées de cette recherche et propose des avancements qui pourraient être faits dans le futur.

CHAPITRE 2

État de la recherche sur les maliciels

2.1 Maliciels

Le terme maliciel désigne une catégorie de logiciels qui sont conçus pour détruire ou endommager des infrastructures informatiques. Cette catégorie regroupe plusieurs types de logiciels nuisibles dont les vers, les virus, les chevaux de Troie, les portes dérobées et les logiciels espions.

2.1.1 Ver informatique

Le terme de ver informatique, *computer worm* en anglais, est tiré d'un livre de science-fiction écrit par Brunner (1975). Dans cet ouvrage, il est question du *tapeworm* qui se propage par le biais d'un réseau informatique et dont les créateurs perdent le contrôle.

C'est ce livre qui inspira les chercheurs Shoch et Hupp (1982) à développer le premier ver de l'histoire de l'informatique. Celui-ci fût développé pour des raisons scientifiques et pratiques. Sa raison d'être était de rassembler les ressources inutilisées d'un laboratoire de recherche pour effectuer des calculs utiles. Ces chercheurs définissent le ver informatique comme suit :

> *A worm is simply a computation which lives on one or more machines.*

Ils décrivent donc un ver comme un calcul informatique qui réside sur un ou plusieurs systèmes informatiques.

Cette définition est correcte mais elle a évolué avec le temps et les utilisations qui ont été faites des vers informatiques. Nazario (2003b) propose la définition suivante :

> *an independently replicating and autonomous infection agent, capable of seeking out new host systems and infecting them via the network.*

C'est à dire un agent d'infection qui se reproduit indépendamment et de manière autonome. De plus, cet agent cherche de nouveaux hôtes et les infecte par le biais d'un réseau informatique.

Les chercheurs américains Weaver *et al.* (2003b) décrivent un ver informatique de la façon suivante :

> *A computer worm is a program that self-propagates across a network exploiting security or policy flaws in widely-used services.*

Pour ces chercheurs, un ver informatique est donc un programme qui se propage par lui-même à travers un réseau en exploitant des failles de politique ou de sécurité dans des services qui sont utilisés à grande échelle.

En réunissant les points communs de ces définitions, on peut décrire un ver informatique comme **un agent indépendant qui évolue dans un réseau en s'introduisant dans les systèmes informatiques**. D'une façon plus générale, on peut parler de ver informatique comme un **agent autonome d'intrusion dans les systèmes informatiques**. Les vers informatiques peuvent effectuer toutes les mêmes opérations qu'un pirate pour gagner l'accès à une ressource informatique. De plus, ils sont automatisés, ce qui les rend beaucoup plus rapides et menaçants.

2.1.2 Virus

Skoudis et Zeltser (2003) décrivent un virus informatique comme suit :

> *A virus is a self replicating piece of code that attaches itself to other programs and usually requires human interaction to propagate.*

Il voit donc les virus comme un petit programme qui se reproduit de façon autonome et qui s'attache à d'autres programmes et requiert l'interaction de l'utilisateur pour se reproduire. Les virus sont donc dépendants des objets auxquels ils s'accrochent. On considère généralement que les actions posées par un virus sont dommageables et violent la politique de sécurité en place sur un système.

Les virus sont présents dans le paysage informatique depuis plusieurs années. Les moyens utilisés pour leur propagation ont beaucoup évolué au cours du temps (Nachenberg, 1997). Nous assistons donc à une course entre les programmeurs de virus et les compagnies qui vendent des logiciels de détection de virus. C'est dans ce contexte de coévolution que certains programmeurs de virus ont ajouté des fonctionnalités de

chiffrement et de polymorphisme aux virus pour les rendre plus difficiles à détecter et à détruire.

Contrairement aux vers informatiques, les virus ont besoin de s'attacher à des fichiers existants sur un système informatique afin de survivre. Les vers informatiques se propagent de manière autonome tandis que les virus se greffent à d'autres fichiers pour leur distribution.

2.1.3 Cheval de Troie

Les chevaux de Troie, *Trojan horses* en anglais, sont définis comme des programmes informatiques qui effectuent des opérations malicieuses à l'insu de l'utilisateur (Shark, 1986). Le nom vient de la légende relatée par Homère dans l'Iliade (Homer, 1998).

> *A trojan horse is a program that appears to have some useful or benign purpose, but really masks some hidden malicious functionality.*

Les chevaux de Troie sont souvent utilisés pour installer des portes dérobées, ou *backdoors*, dans les systèmes informatiques (firew0rker, 2004) . Ces programmes peuvent aussi être utilisés pour récolter des informations sur les utilisateurs, mémoriser des mots de passes saisis, copier des données sensibles, etc. À la différence d'un virus, le cheval de Troie est souvent installé par un utilisateur malveillant et il ne cherche pas à se multiplier ou à se propager.

2.1.4 Porte dérobée

> *A backdoor is a program that allows attackers to bypass normal security controls on a system, gaining access on the attacker's own terms.*

Une porte dérobée, *backdoor* en anglais, est un terme informatique qui décrit un mécanisme qui assure un accès à un système (Skoudis et Zeltser, 2003). Une porte dérobée permet de contourner les mécanismes d'authentification réguliers pour obtenir un accès distant au système. Il est possible d'installer une porte dérobée comme un programme indépendant ou de modifier un programme existant et d'y ajouter certaines fonctionnalités (firew0rker, 2004; klog, 2000).

Comme leur nom l'indique, les portes dérobées doivent être difficiles à détecter pour qu'un attaquant garde accès à un système le plus longtemps possible même après

des mises à jour ou l'installation de nouveaux logiciels.

2.1.5 Logiciel espion

Les logiciels espions, *spyware* en anglais, désignent une catégorie de logiciels malicieux qui sont construits pour surveiller et rapporter à une tierce personne les activités des utilisateurs (Stafford et Urbaczewski, 2004). Les personnes qui installent un logiciel espion sur un système sont rénumérées d'une petite somme d'argent par la compagnie récoltant les informations. Si un attaquant installe un logiciel espion sur plusieurs milliers de systèmes, il peut gagner d'importantes sommes d'argent en peu de temps. Les gains potentiels importants font que ce type de maliciels est une menace pour les utilisateurs et les administrateurs réseau.

Les logiciels espions peuvent être utilisés légalement par une compagnie qui désire mieux connaître les habitudes de consommation de ses clients. Plusieurs *spyware* sont installés quand un utilisateur télécharge un logiciel gratuit. La condition d'utilisation de ce logiciel est d'aussi installer un logiciel espion qui permettra aux créateurs de financer leur développement. Les logiciels espions peuvent aussi être installés illégalement par un attaquant qui réussi à prendre le contrôle d'un système. Il utilisera alors un logiciel espion pour connaître les touches tapées par sa victime ou examiner le contenu de son disque dur.

Les logiciels espions, même s'ils sont installés légalement, sont nuisibles pour l'utilisateur, premièrement, en raison des risques liés à l'accès à des information privées comme les mots de passe et les habitudes de consommation. De plus, les logiciels espions doivent communiquer avec un serveur tiers pour envoyer les informations capturées. Ces transactions demandent de la bande passante et des ressources informatiques que l'utilisateur n'est pas nécessairement prêt à sacrifier. Finalement, les ressources de l'ordinateur, notamment la mémoire et le temps de processeur, sont utilisées par le logiciel espion et peuvent diminuer les performances des systèmes infectés.

2.2 Historique du maliciel

Cette section décrit quelques-unes des plus grandes épidémies de maliciels qui ont été observées sur l'Internet. La section 2.2.1 décrit les premières expériences sur les

vers informatiques tandis que la section 2.2.2 décrit la première épidémie de vers informatiques sur l'Internet. La section 2.2.3 décrit le fonctionnement du ver *Code Red*. La section 2.2.4 décrit le ver *Slammer* et la section 2.2.5 décrit le fonctionnement du *PhatBot*, un logiciel malicieux souvent observé sur les réseaux contemporains.

2.2.1 Ver "worm program"

C'est en 1982 que le concept de ver informatique fut mis pour la première fois à l'essai dans les laboratoires de Xerox à Palo Alto (Shoch et Hupp, 1982). Ce programme informatique a été conçu dans le cadre des premières expériences sur l'informatique distribuée. Le ver était destiné à faire usage des ressources inutilisées des ordinateurs du laboratoire pour effectuer des calculs. Il était composé de trois sections indépendantes :

- Code d'initialisation qui est utilisé lors du démarrage sur la première machine.
- Partie d'initialisation quand le ver s'installe sur une machine subséquente.
- Le programme principal à exécuter.

Pour effectuer une propagation "polie" dans le réseau ethernet du laboratoire, un petit protocole est écrit pour demander à l'ordinateur s'il est inactif. Si c'est le cas, on demande à celui-ci de redémarrer et de charger le programme du ver lors de son démarrage. Pour ne pas étrangler le réseau informatique, il n'y que la tête du ver qui cherche à se propager, c'est à dire que tous les autres systèmes "infectés" par le ver sont uniquement employés pour effectuer des calculs.

Dès les premières expériences, la nature indépendante et envahissante du ver informatique a laissé paraître les dangers de son utilisation. Les chercheurs ont laissé s'exécuter un ver pendant la nuit sur tout le réseau de leur laboratoire de Palo Alto. Une des versions du ver s'est corrompue lors de son transfert sur le réseau. Quand un ordinateur essayait de charger le virus, une erreur survenait et l'ordinateur s'éteignait. Vu que le ver n'avait pas réussi à se propager, il continuait à chercher d'autres systèmes inactifs pour s'installer. Le matin, les chercheurs ont trouvé plus d'une centaine d'ordinateurs défectueux dispersés dans leurs locaux et ils ont du redémarrer manuellement.

2.2.2 Ver Morris

C'est le 2 novembre 1988 que la première épidémie de vers fut observée sur l'Internet. C'est à cette date qu'un ver, qui fut plus tard baptisé du nom de son auteur,

fit son apparition (voir l'article de Spafford (1989) pour une description détaillée).

Le ver Morris prend pour cible les systèmes Unix branchés à Internet. Ce ver est très intéressant, non seulement par le fait qu'il est apparu très tôt mais aussi par la quantité de vecteurs d'attaques différents qu'il utilise pour compromettre et infecter ses victimes.

Le ver Morris exploite plusieurs failles qui étaient présentes sur les systèmes Unix des années 80. Il exploite une faille par débordement de tampon dans le service *fingerd* et une fonction de maintenance (*debug*) présente dans le serveur SMTP *sendmail*. En plus d'exploiter des failles de nature technique dans certains logiciels, le ver Morris exploite aussi des faiblesses crées par les utilisateurs. Premièrement, le ver essaie de déduire les mots de passes des utilisateurs du système attaqué. De plus, une fois qu'une machine est compromise, le ver essaie d'utiliser les relations de confiance entre les hôtes (*rsh*) pour continuer sa propagation.

Pour signifier à son auteur qu'il a infecté un nouveau système, le ver envoie un paquet UDP à un ordinateur connecté sur le réseau de l'Université Berkeley aux États-Unis. Ce mécanisme de communication rend la détection des machines infectées assez facile. Quand le ver Morris réussit à infecter une nouvelle machine, il se connecte sur l'ordinateur attaquant pour télécharger une nouvelle version du ver et, par la suite, continuer sa propagation à partir de l'ordinateur nouvellement infecté.

Le ver Morris était le premier à apparaître sur l'Internet. Vu sa nature unique, plusieurs chercheurs l'ont analysé. Une des conclusions commune de ces chercheurs est que ce maliciel est mal programmé. Plusieurs variables y sont déclarées mais jamais utilisées, des sections entières de code ne sont jamais exécutées, etc. On pense que le ver a été libéré sur Internet de façon accidentelle alors qu'il était encore en développement. Ces observations laissent croire que, malgré le fait que les dommages causés par le ver Morris furent importants, les conséquences auraient pu être pires.

2.2.3 Ver Code Red

Selon les statistiques collectées lors de l'infection du ver CodeRed (Moore *et al.*, 2002), plus de 359 000 ordinateurs auraient été touchées en moins de 14 heures par ce maliciel. Les dégâts économiques causés par ce ver furent très importants, ils sont estimés à près de 2.6 milliards de dollars américains.

Le ver CodeRed exploite une faille de sécurité dans le serveur web *Internet In-*

formation Services (IIS) de Microsoft. Cette faille par débordement de tampon a été découverte et publiée par Ryan Hassell de la compagnie eEye en juin 2001 (Riley Hassell, 2001). Huit jours après la découverte de cette vulnérabilité, soit le 26 juin 2001, Microsoft a émis une rustine pour corriger le problème (Microsoft Security Response Team, 2001). Ce n'est que 25 jours plus tard, le 12 juillet 2001, que CodeRed fit sa première apparition sur l'Internet.

Contrairement au ver Slammer qui se propage à l'aide de son vecteur d'attaque (voir la section 2.2.4), CodeRed propage sa charge active une fois qu'il a infecté une victime. Lors de l'activation de sa charge active, il effectue les opérations suivantes selon la date de l'ordinateur qu'il vient d'infecter :

- Si la date du système est entre le 1 et 19 du mois, le ver génère une liste d'adresses IP aléatoire et tente de les infecter.
- Entre le 20 et le 28 de chaque mois, les ordinateurs infectés par le ver sont utilisés pour lancer une attaque par déni de service contre le site web `http ://www1.whitehouse.gov`.
- Après le 28 du mois, le ver tombe en hibernation.

La première version de CodeRed était moins dangereuse parce qu'elle contenait plusieurs problèmes. Premièrement, le générateur de nombres pseudo aléatoires qui est utilisé pour identifier des nouvelles victimes est toujours initialisé avec la même valeur. Cette erreur fait en sorte que chaque instance du ver essaie d'infecter exactement la même liste de cibles. De plus, le ver reste en mémoire une fois qu'il a infecté une victime. Un simple redémarrage d'une machine infectée permet donc de la désinfecter. C'est pour pallier à ces faiblesses qu'une deuxième version du ver CodeRed a fait son apparition 6 jours après la première. C'est cette dernière version qui fut la plus virulente et qui fit le plus de dégâts en infectant 359 000 ordinateurs en moins de 14 heures.

La deuxième version de CodeRed (baptisée CodeRedII) exploite la même faille dans le serveur IIS que CodeRed. Par contre, cette version installe une porte arrière sur le serveur pour garantir à l'auteur un accès à cet ordinateur. De plus, une journée après l'infection, le ver redémarre le système sur lequel il se trouve. Vu que le ver n'est plus stocké en mémoire mais sur le disque, le fait de redémarrer le serveur n'est pas dommageable pour le ver. Le fait que l'ordinateur redémarre seulement une journée après avoir été infecté rend l'identification des victimes beaucoup plus difficile parce que le logiciel malicieux n'est pas activé immédiatement.

2.2.4 Ver Slammer

Le ver Slammer, parfois appelé Saphine ou SQL Slammer, a fait son apparition sur Internet en janvier 2003 (Moore *et al.*, 2003a). Ce ver est remarquable par la vitesse à laquelle il se propage. En effet, en moins de dix minutes, il a réussi à infecter près de 90% de ses victimes potentielles. La propagation de ce ver fut tellement rapide que plusieurs segments de l'Internet furent débordés par le trafic généré par les balayages du ver. Les conséquences économiques furent significatives : vols annulés, guichets automatiques hors d'usage, etc.

Slammer exploite une faille de débordement de tampon dans le logiciel de base de données *SQL Server* de Microsoft (Microsoft Security Response Team, 2003). Les détails de cette faille de sécurité ont été publiés le 11 juillet 2002 par David Litchfield Litchfield (2002). Cette faille était donc connue depuis cinq mois avant l'apparition de Slammer. Malgré la disponibilité de rustines pour corriger cette faille, plus de 75 000 hôtes ont été infectésonts.

Heureusement, Slammer n'a pas de charge dommageable, les seuls troubles causés par celui-ci sont dus à la rapidité avec laquelle il se propage. Ce ver choisit aléatoirement ses victimes à l'aide d'un générateur de nombres pseudo aléatoires. Une fois l'adresse IP d'une cible générée, le ver envoie un seul paquet UDP qui exploite le débordement de tampon. Le vers n'a pas besoin d'attendre de réponse à son attaque, accélérant ainsi l'infection de nouvelles victimes. On estime que pour une connexion de 100 mégabits par seconde, le vers peut envoyer environ 26 000 paquets malveillants par seconde. Ceci explique la grande congestion des réseaux causée par ce programme.

Malgré sa propagation extrêmement rapide, l'analyse du ver Slammer (Moore *et al.*, 2003a) montre que certaines erreurs ont été commises lors de sa conception. Par exemple, le générateur de nombres pseudo-aléatoires utilisé pour trouver de nouvelles cibles comportait une erreur qui fait en sorte que certaines adresses reviennent plus souvent que d'autres. On peut donc conclure que la propagation de ce vers aurait pu être encore plus rapide si une telle erreur avait été corrigée. De plus, Slammer ne transporte aucune charge dommageable. Les conséquences auraient pu être beaucoup plus dramatiques si, par exemple, il avait effacé toutes les données contenues dans les bases de données qu'il a compromises.

2.2.5 Maliciel PhatBot

Le programme *PhatBot* est un maliciel plutôt récent. C'est pourquoi très peu d'articles scientifiques y font référence. Le PhatBot est un logiciel qui peut être installé sur un système infecté par un attaquant. Il comporte des modules de communication utilisés pour centraliser le contrôle de systèmes infectés (*botnets*). Il contient aussi des modules d'exploitation de failles de sécurité qui sont utilisés pour infecter de nouveaux systèmes. La phase de propagation peut être effectuée manuellement par la personne qui contrôle le programme ("*0wner*") ou automatiquement. Finalement, ce logiciel contient des caractéristiques d'un logiciel espion parce qu'il peut lire les touches tapées par un utilisateur (*keylogger*) ou analyser les informations contenues sur le disque dur à la recherche d'information intéressantes. Concrètement, ces recherches sur le disque ou les frappes de l'utilisateur sont faites à l'aide d'expressions régulières. L'attaquant peut donc autant chercher des informations bancaires que des numéros de série de jeux installés chez la victime.

Le maliciel *PhatBot* est programmé en C++ et est une évolution du maliciel Agobot. Sa construction orientée objet le rend facile à maintenir et à améliorer. Ce logiciel contient aussi des fonctions de mise à jour. Un attaquant peut donc créer un module d'exploitation pour une nouvelle faille de sécurité et l'envoyer à tous les systèmes déjà infectés.

PhatBot est un bon exemple de l'évolution des logiciels malicieux. On ne peut plus parler de ver, de porte dérobée ou de logiciel espion pour le caractériser parce qu'il comporte des caractéristiques de toutes ces catégories de maliciels.

2.2.6 Virus Melissa

Le virus Melissa est un maliciel qui se propage uniquement par la voie des courriels. Il a fait son apparition sur l'Internet le 26 mars 1999. Ce virus n'exploite pas de failles de sécurité proprement dites mais plutôt une erreur de conception dans les applications de la suite Office de Microsoft. Ces applications laissent s'exécuter automatiquement le code Visual Basic for Application (VBA) quand un utilisateur ouvre un document. Le virus Melissa est donc une série d'instructions VBA emmagasinées dans un fichier Word. Ces instructions font en sorte que, quand un utilisateur ouvre le fichier infecté, un courriel contenant une copie du fichier est envoyé aux cinquante premières personnes qui se trouvent dans le carnet d'adresses de la victime.

Certaines statistiques (Garber, 1999) affirment que plus de 1.2 millions d'ordinateurs ont été infectés par le virus Melissa. En plus de remplir les boîtes de courriels des utilisateurs, plusieurs serveurs n'ont pu soutenir le grand nombre de transactions et ont arrêté de fonctionner. Les coûts de réparation ont été estimés à quelques centaines de millions de dollars américains pour l'Amérique du Nord seulement.

L'originalité du virus Melissa réside dans son mécanisme de propagation. C'est en effet le premier logiciel malicieux à se propager par courriel. De plus, le logiciel utilise de fausses informations pour inciter les utilisateurs à ouvrir le fichier. En effet, les sujets des courriels envoyés par Melissa sont conçus pour attirer l'attention, par exemple, "I love you .." avec le nom de l'expéditeur qui a déjà été infecté ou "Here is the file we talked about".

2.2.7 Ver Blaster

Le ver Blaster est un autre logiciel malicieux qui s'attaque aux systèmes Windows de Microsoft. Plus particulièrement, Blaster exploite une faille de sécurité dans le systèmes Distributed Component Object Model (DCOM). Le système DCOM est un ensemble d'interfaces de programmation permettant à différents systèmes d'interagir en échangeant des informations, des programmes ou des fichiers. Cette faille a été découverte et documentée par le groupe Last Stage of Delirium (LSD) le 16 juillet 2003. Un mois plus tard, le 11 août 2003, le ver Blaster fait son apparition (Nazario, 2003a).

Le comportement de ce ver est intéressant parce qu'il utilise un mécanisme de décision probabiliste pour décider quelles actions prendre. Lors de sa phase de balayage, le ver commence par parcourir les 254 systèmes qui se trouvent dans son sous-réseau IP. Par la suite, il choisit de balayer un sous-réseau de 256 adresses (/24) dans son sous-réseau de 65536 (/16) adresses avec une probabilité de 40% et un sous-réseau de 256 adresses (/24) choisi aléatoirement avec une probabilité de 60%. Ce logiciel utilise aussi les probabilités quand vient le temps d'exploiter la faille de sécurité DCOM ; 80% des tentatives de pénétration sont effectuées en considérant que la victime est un système Windows XP et 20% des tentatives sont faites comme si la victime est un système Windows 2000. La complexité des paramètres de décision de ce ver nous laisse croire que certains choix de conception ont été faits pour maximiser sa performance. Ce ver est le premier à montrer un processus décisionnel aussi

complexe.

Quand Blaster réussit à infecter une nouvelle victime, il transfert son exécutable nommé `msblast.exe` à l'aide du protocole TFTP, donc sur une couche de transport User Datagram Protocol (UDP). En plus de continuer sa propagation, ce ver a une charge active qui lance une attaque par déni de service contre le site `windowsupdate.com` le 16 août.

2.3 Taxonomie des vers informatiques

Tout comme les attaques d'un pirate, le comportement d'un maliciel peut être disséqué en plusieurs parties distinctes. Cette section s'attarde à quelques taxonomies de vers informatiques. Nous portons une attention particulière aux vers informatiques parce qu'ils sont les maliciels qui ont le plus d'indépendance dans leur propagation. Des taxonomies existent pour les virus et les chevaux de Troie mais nous jugeons que ce sont les caractéristiques des vers informatiques qui sont les plus complètes et variées. Plusieurs chercheurs ont proposé des taxonomies différentes des logiciels malicieux et plus précisément des vers informatiques. Les chercheurs de la compagnie CrimeLabs (Nazario *et al.*, 2001) divisent le comportement des vers informatiques en six parties différentes présentées au tableau 2.1.

TABLEAU 2.1 Taxonomie de CrimeLabs

Reconnaissance	Identification de cibles selon différents critères.
Attaque	Méthode utilisée par un noeud d'un ver pour gagner l'accès à un nouveau système.
Interface de commande	Moyen utilisé pour contrôler les noeuds du ver.
Communication	Transfert d'informations entre les noeuds d'un ver.
Intelligence	Connaissance qu'un ver a des autres noeuds infectés sur le réseau.
Attaques inutilisées	Un ver peut utiliser plusieurs vecteurs d'attaque, il est possible que toutes ces capacités ne soient pas utilisées.

Quand on parle de ver informatique ou de maliciel en général, un noeud infecté correspond à un système. Le ver en général désigne l'ensemble des systèmes infectés.

De leurs côté, Weaver *et al.* (2003b) différencient les vers selon cinq propriétés. Les caractéristiques de la taxonomie apportée par les chercheurs de Berkeley sont présentées dans le tableau 2.2.

TABLEAU 2.2 Taxonomie de Weaver

Découverte et sélection des cibles	Identification de nouvelles victimes qui sont vulnérables aux attaques du ver
Mécanisme de propagation	Souvent une simple copie du fichier exécutable du ver d'une machine infectée vers une autre.
Mécanisme d'activation	Peut être un compte à rebours mais souvent l'activation est faite tout de suite après l'infection.
Charge active	Actions qui sont exécutées après l'infection et qui ne concernent pas la propagation ou la survie du logiciel malicieux.
Motivations et utilisation de l'auteur.	Les conséquences d'un ver changent beaucoup en fonction des motivations de leur auteur.

Un chercheur indépendant en sécurité nommé Zalewski (2000) énumère sept propriétés qu'un ver devrait avoir. Celles-ci sont présentées dans le tableau 2.3.

TABLEAU 2.3 Taxonomie de Zalewski

Portabilité	Le programme du ver informatique doit être en mesure de s'exécuter sur le plus de systèmes possibles, indépendamment de le version du système d'exploitation et des logiciels installés.
Invisibilité	Difficulté de détection par un utilisateur ou un administrateur.
Indépendance	Le ver doit se propager de façon indépendante, avec une réserve de modules d'exploitation.
Apprentissage	Apprendre de nouvelles stratégies d'attaque à l'aide de communications
Intégrité	La structure du ver doit être difficile à retracer, modifier et exterminer.
Polymorphisme	Ne pas avoir de signature détectable
Utilisable	Remplir les objectifs fixés par l'auteur.

Finalement, Todd (2003) sépare les caractéristiques d'un ver selon leurs utilités. Il identifie quatre composantes nécessaires pour qu'un ver soit fonctionnel, celles-ci sont présentées dans le tableau 2.4. En plus des quatre caractéristiques de base, les maliciels

peuvent avoir des fonctionnalitées de défense, c'est à dire qu'ils peuvent prendre des actions pour être plus difficiles à détecter et arrêter. La taxonomie de Todd tient aussi compte de l'interface de commande qui peut être utilisé par l'attaquant pour partager l'information collectée par les instances malicieuses.

TABLEAU 2.4 Taxonomie de Todd

Autonomie	Par définition, un ver est fait pour pouvoir exister sans intervention extérieur.
Multiplication	Le code est copié, de façon normale ou par polymorphisme.
Reconnaissance	Identification de nouvelles cibles pour se propager.
Capacités d'attaque	Le ver utilise une faille connue (débordement de tampon ou autre). Une fois la faille exploitée, il exécute des instructions pour se reproduire.

2.4 Modélisation des vers informatiques

Plusieurs caractéristiques des maliciels ressemblent beaucoup aux virus biologiques. C'est pourquoi les modèles pour prédire la rapidité de propagation et le nombre d'hôtes infectés se basent sur les théories épidémiologiques développées en biologie. Le modèle le plus simple utilisé est le *modèle épidémiologique classique*. Des modèles plus évolués ont été développés par Kermack et McKendrick (1927) pour tenir compte des systèmes qui sont désinfectés et par Zou *et al.* (2002) pour tenir compte de la variation du taux d'infection.

Le modèle épidémiologique classique (Frauenthal, 1980) a été développé pour décrire la propagation de maladies auprès d'une population. Ce modèle est facilement applicable à l'informatique. Ce modèle considère que les systèmes informatiques peuvent être dans deux états lors d'une épidémie de maliciels : *susceptible*, c'est à dire vulnérable aux attaques et *infecté*, c'est à dire que le système a déjà subi les attaques d'un logiciel malicieux, qu'il est maintenant infecté et qu'il tente d'infecter des nouvelles victimes. La seule transition d'état possible pour un système est de passer de l'état *susceptible* à l'état *infecté*. L'équation de base du modèle épidémiologique classique est la suivante :

$$\frac{dJ(t)}{dt} = \beta J(t)[N - J(t)] \tag{2.1}$$

où $J(t)$ représente le nombre d'hôtes infectés au temps t, N la taille totale de la population et β le taux d'infection.

Posons $a(t) = J(t)/N$ comme la fraction de la population qui est infectée au temps t. En divisant les deux côtés de l'équation 2.1 par N^2, on obtient l'équation suivante utilisée par (Staniford *et al.*, 2002) :

$$\frac{d\,a(t)}{d\,t} = k\,a(t)[1 - a(t)] \tag{2.2}$$

Avec ce modèle, on peut modéliser la propagation d'un ver informatique sur l'Internet avec une précision intéressante. La courbe de croissance de la population infectée est presque exponentielle jusqu'à ce qu'environ 85% de la population soit infectée. Après ce seuil, la croissance ralentit considérablement.

Le modèle Kermack-Mckendrick (Frauenthal, 1980) améliore le modèle classique en considérant le fait que certains systèmes infectés peuvent être soit réparés ou simplement éteints. Ce modèle ajoute donc un état possible pour tous les systèmes, l'état *retiré*. Dans ce modèle, les systèmes peuvent passer de l'état *susceptible* à *infecté* et de l'état *infecté* à l'état *retiré*. Avec $J(t)$ le nombre total de systèmes infectés au temps t, $I(t)$ le nombre de systèmes couramment infectés et $R(t)$ le nombre de systèmes qui ont été retirés, on obtient l'équation suivante :

$$J(t) = I(t) + R(t) \tag{2.3}$$

Des chercheurs de l'université du Massachusetts (Zou *et al.*, 2002) améliorent encore une fois le modèle en considérant la variation du taux de propagation β. En effet, sur des gros réseaux, si plusieurs machines balayent de façon intensive, il se créé des congestions qui font en sorte que les performances de balayage sont diminuées. C'est pour tenir compte de ces congestions que les auteurs définissent $\beta(t)$ qui est le taux de propagation au temps t. En plus du taux de propagation variable, les auteurs introduisent $Q(t)$ qui est le nombre de systèmes qui sont susceptibles à l'infection mais qui sont retirés de l'environnement. Le nouveau modèle est appelé "modèle à deux facteurs" et est représenté par l'équation suivante :

$$\frac{dI(t)}{dt} = \beta(t)[N - R(t) - I(t) - Q(t)]I(t) - \frac{dR(t)}{dt} \tag{2.4}$$

Les modèles utilisés pour prédire la vitesse de propagation d'un maliciel ainsi que le nombre de systèmes qui seront infectés ont aussi leurs faiblesses. En effet, aucun de

ces modèles ne tient compte de la topologie des réseaux et des équipements défensifs installés. Ils sont aussi peu utils pour prédire l'impact local d'une épidémie, ils sont uniquement utilisables pour peindre une idée générale des conséquences possibles.

2.5 Simulation des infections de vers

Les outils de simulation de propagation de logiciels malicieux ont été créées pour pallier aux faiblesses des modèles théoriques. L'outil développé par des chercheurs de l'institut des technologies de Suisse (Wagner *et al.*, 2003) est un simulateur qui simule le nombre de systèmes infectés dans un réseau en fonction du temps, de la bande passante et du protocole de transport utilisé par le maliciel. Le simulateur se base sur les statistiques récoltées sur les réseaux pair à pair *Gnutella* et *Napster* pour créer un réseau de réseaux regroupant les systèmes en fonction de leur bande passante. Le simulateur tient aussi compte de la différence entre les protocoles de transport UDP et TCP puisque TCP demande la transmission de plus de données et impose souvent des délais plus longs que UDP lors de transferts d'informations.

Le simulateur utilise l'équation suivante pour déterminer la probabilité qu'un système devienne infecté à chaque itération.

$$P(i) = \frac{|s_v| - |s_i|}{|s_p|} \tag{2.5}$$

La variable s_v représente le nombre de systèmes vulnérables, s_i le nombre de systèmes infectés et s_p le nombre total de systèmes présents. Malgré que ce simulateur considère la topologie des réseaux par rapport à la bande passante disponible pour chacun d'eux et leur interconnexion, il ne tient pas compte de la distribution des systèmes vulnérables sur chacun des réseaux. En effet, le simulateur considère que les systèmes vulnérables sont distribués uniformément dans tous les sous-réseaux, ce qui ne représente pas bien la réalité de l'Internet où l'on risque de trouver des sous-réseaux où tous les systèmes sont vulnérables et d'autres qui utilisent d'autres logiciels et qui sont donc immunisés.

Le simulateur a été implanté par les chercheurs et les résultats obtenus sont intéressants. Le simulateur a été utilisé pour simuler le comportement des vers Slammer et CodeRed. Le fait qu'il utilise un modèle de l'Internet précis est l'atout principal de son efficacité. Par contre, les chercheurs n'ont pas publié de résultats sur la simu-

lation du comportement de vers qui n'ont pas encore été observés sur l'Internet.

2.6 Évolution future des maliciels

Il est important d'identifier quelles seront les nouvelles tendances utilisées par les créateurs de maliciels. De telles connaissances permetteront de préparer les défenses à l'avance et d'avoir une démarche proactive. Cette section décrit certaines évolutions envisageables pour les logiciels malicieux en fonction de leurs caractéristiques.

2.6.1 Identification des cibles

La phase d'identification de nouvelles cibles par un ver est la phase la plus bruyante et donc la plus facile à détecter. En effet, un système de détection d'intrusion ou un administrateur compétent identifiera rapidement un problème si un nombre croissant de requêtes sont effectuées vers des adresses qui n'existent pas (Riordan *et al.*, 2005).

Certains chercheurs prévoient l'apparition d'un nouveau type de ver qui utilise des techniques nouvelles pour identifier leurs cibles. Par exemple, les chercheurs de l'Université du Massachusetts décrivent les "routing worms" (Zou *et al.*, 2005). Ces programmes malicieux utilisent les informations contenues dans les tables de routage pour seulement s'attaquer à des cibles qui sont fonctionnelles. En plus de rendre le progression du ver plus difficile à détecter, le ver pourrait réduire son espace de recherche de 71% puisqu'il est estimé que seulement 29% des blocs d'adresses sont fréquemment utilisés sur l'Internet.

Au lieu d'effectuer des requêtes directement vers des cibles potentielles au niveau de la couche réseau, les vers peuvent aussi utiliser les informations des serveurs de nom DNS (Hanhua, 2005). Grâce à ce procédé, le ver augmente ses chances d'effectuer une requête vers un système existant et réduit les chances de détection. Par contre, il est possible d'imposer un délai entre les requêtes effectuées vers un serveur de nom, ralentissant ainsi fortement la vitesse de progression de ce type de maliciel.

Chaque noeud d'un ver n'est pas obligé d'effectuer une phase de reconnaissance. L'auteur d'un ver pourrait effectuer un balayage complet de l'Internet avant de lancer son ver et d'y incorporer une liste complète des cibles potentiellement vulnérables (Staniford *et al.*, 2002). Considérant que la phase de découverte de nouvelles cibles est l'une des plus longue lors de la propagation d'un ver, une liste pré-compilée de

cibles rendrait la propagation du ver très rapide, d'où l'appellation *flashworm*.

Finalement, un ver peut tirer profit des méta-informations qui sont utilisées par différents services (Weaver *et al.*, 2003b). Par exemple, les réseaux de jeux en ligne gardent souvent une liste de tous les serveurs disponibles. Un ver pourrait consulter cette liste pour s'attaquer uniquement à ces serveurs vulnérables. Les services de partage de fichiers de Windows fournissent des informations sur tous les répertoires partagés d'un groupe de travail. Ces informations sont aussi très favorables pour la propagation d'un ver (Weaver *et al.*, 2003a).

Une autre technique pour découvrir des victimes qui pourrait être observée dans le futur est la propagation par "saut" (Nazario, 2003b). Au lieu de permettre à tous les noeuds d'un ver de détecter des nouvelles cibles, seulement la "tête" du ver peut balayer le réseau et infecter des nouveaux systèmes. Ce type d'attaque générerait beaucoup moins de trafic qu'un ver classique, ce qui le rend plus difficile à détecter. De plus, cette technique pourrait être utilisée par des vers qui ont une cible précise. Dans ce cas, le ver se propage jusqu'à ce qu'il atteigne sa cible et après il arrête complètement sa propagation.

2.6.2 Communication

La communication entre les noeuds d'un ver est importante pour pouvoir propager des mises à jour du programme, coordonner une nouvelle phase d'attaque ou simplement utiliser les systèmes qui ont été infectés. La communication des vers peut être chiffrée pour assurer l'intégrité des mises à jour qui sont envoyées aux noeuds (Nazario *et al.*, 2001). De plus, une partie de la charge active d'un ver peut être chiffrée pour rendre sa détection plus difficile lors des transactions sur le réseau(Nachenberg, 1997).

La topologie des réseaux de communication entre les vers a été longuement étudiée. Certains proposent des schémas semblables au groupe de guérilla (Nazario *et al.*, 2001) pendant que d'autres favorisent un réseau de pair à pair comme ceux utilisés pour le partage de fichiers (Wiley, 2002). L'organisation de guérilla (Guevara, 1961) impose aux vers de s'organiser par petits groupes de systèmes infectés. De cette façon, la détection d'une cellule ne compromet pas l'existence du ver dans son entier. D'un autre côté, les réseaux pair à pair permettent d'augmenter la fiabilité des transactions entre les noeuds sans l'existence d'une structure centrale. Cette architecture n'introduit pas

de point de faille central en cas de détection d'un noeud du ver.

2.6.3 Infection

Avec l'évolution de la technologie et des services offerts par les réseaux informatiques, de nouveaux vecteurs d'infection apparaissent. Par exemple, une panoplie de techniques d'exploitation ont vu le jour pour les applications Web et les engins de recherche (Scott et Sharp, 2002; Ben-Artzi et Strormberg, 2003; SPI Laboratories, 2002).

Très peu de maliciels se sont attaqués aux applications Web jusqu'à date. Ces applications sont très exposées sur l'Internet et plusieurs type de failles de sécurité existent dans ces applications. C'est pourquoi, il ne serait pas surprenant de voir émerger un nouveau type de logiciel malicieux qui s'attaque spécialement aux applications Web (CGI Security, 2002).

En plus d'avoir un terrain très fertile pour l'infection de fichier et la diffusion d'information malicieuse, les maliciels qui s'attaquent aux applications Web peuvent utiliser des nouveaux moyens pour découvrir leurs cibles. Par exemple, ils peuvent utiliser des moteurs de recherche public comme *Google* ou même implanter un algorithme de recherche des liens d'une page Web pour trouver certains logiciels vulnérables.

Une autre technique d'infection qu'on pourrait voir apparaître est l'utilisation d'araignées. La plupart des engins de recherche modernes utilisent des processus automatisés qui parcourent les pages Web dans le but d'accumuler de l'information. Ces processus sont parfois appelés robots ou araignées parce qu'ils suivent les liens entre les pages Web comme une araignée suit les fils de sa toile.

Vu que les araignées des engins de recherche suivent les liens affichés sur les pages Web sans connaissance et vérification préalable, il est possible de se servir de ceux-ci pour effectuer des attaques Web (Zalewski, 2001). En mettant des liens vers des sites avec des arguments malveillants, les araignées effectuent les attaques vers les autres services. L'attaquant n'a qu'à établir une liste de liens vers ses victimes avec des arguments qui exploitent des failles de sécurité connues et c'est l'araignée qui effectue le reste du travail.

Pour rendre la détection d'un ver plus difficile, certains chercheurs prévoient l'apparition de vers polymorphes. C'est à dire un ver qui peut prendre plusieurs formes pour se déplacer sur le réseau. Certains outils qui effectuent de modifications poly-

morphiques de fichiers binaires sont déjà disponibles sur l'Internet (Zovi, 2001; Moore, 2003) et utilisent le polymorphisme. Aucun ver polymorphe n'a encore été observé sur l'Internet. La détection de ce type ver par les système de détection d'intrusion basés sur les signatures serait très difficile en raison de la nature changeante des données qui se propagent entre les systèmes.

2.6.4 IPv6

Le nouveau protocole de l'Internet, IPv6 (Hauser, 2006; Hagino, 2006) amène de nouvelles considérations sur la propagation des maliciels dans les réseaux. Premièrement, l'élargissement de l'espace d'adressage rend la propagation de maliciels qui cherchent leurs victime de façon aléatoire beaucoup plus improbable. En effet, des milliards d'adresses ne seront pas utilisées dans les années à venir, ce qui fait en sorte qu'un ver passera la grande majorité de son temps à essayer d'attaquer des systèmes inexistants. Deuxièmement, l'IPv6 présente de nouveaux mécanismes pour découvrir les systèmes qui sont sur le même segment réseau. Ce type de mécanisme pourrait faciliter de beaucoup la découverte de nouvelles cibles par un logiciel malicieux qui n'aurait plus à effectuer des requêtes vers des systèmes inexistants.

Il est aussi à noter que, présentement, certains systèmes sont très bien défendus au niveau de l'IPv4 mais qu'ils sont aussi accessibles par l'IPv6 avec des défenses moins élaborées. Cette différence de niveau sécurité est liée au manque d'informations de plusieurs administrateurs face à l'IPv6. Cette ouverture pourrait être exploitée dans un future proche par un programme malicieux.

2.7 Défense

Il est évident que la meilleure défense contre les vers informatiques est de garder les systèmes à jour en appliquant les rustines de sécurité dès qu'elles sont publiées et en configurant de façon sécuritaire tous les équipements déployés. Malheureusement, les chiffres démontrent (Rescorla, 2003) que les administrateurs de systèmes sont souvent surchargés ou craintifs et n'appliquent pas assez rapidement ces rustines. C'est pourquoi plusieurs chercheurs se sont penchés sur le développement de technologies spécifiquement pour détecter et bloquer la progression de logiciels malicieux

2.7.1 Logiciels anti-virus

Un anti-virus est un programme qui a pour but de détecter, stopper et éliminer les virus et autres logiciels malicieux. Comme pour les systèmes de détection d'intrusion, les anti-virus utilisent deux paradigmes différents pour détecter les maliciels. Le premier paradigme est l'identification de signatures malicieuses dans les fichiers traités. Le deuxième paradigme se base sur la détection de comportement malicieux lors de l'exécution de programmes.

Les maliciels ont évolué en contact avec les anti-virus (Nachenberg, 1997). Sachant que les anti-virus gardent un dictionnaire des signatures de fichiers malicieux, les auteurs de maliciels ont créés des programmes qui se modifient eux-même afin d'éviter la détection et l'éradication.

Le logiciel anti-virus constitue une bonne défense contre beaucoup de maliciels. Par contre, étant donné qu'ils s'exécutent sur l'hôte, ils peuvent être désactivés quand un maliciel infecte sa victime. C'est pourquoi des mesures externes doivent aussi être mises en oeuvre afin de détecter les attaques réussies.

2.7.2 Détection d'intrusion

Le concept de détection d'intrusion dans les réseaux est assez vieux. Dans un article publié à la fin des années 80, Denning présente les fondements d'un système expert pour détecter les cas de piratage et autres abus de ressources informatiques (Denning, 1987). L'hypothèse de ses travaux est qu'on peut détecter les comportements malicieux grâce à l'analyse automatisée des fichiers de journalisation d'un système informatique.

On peut différencier deux grands types de systèmes de détection d'intrusion : les systèmes basés sur les signatures et les systèmes basés sur la détection d'anomalie.

Un système de *détection d'intrusion par signatures*, aussi connu sous le nom de détection d'abus, analyse le trafic d'un réseau à la recherche de signatures connues. Les signatures sont compilées par un analyste et représentent des attaques connues que l'on doit détecter et arrêter si possible. Le système le plus connu de détection d'intrusions par identification de signatures s'appelle *snort* (SNORT, 2002).

Les systèmes de détection d'intrusion basés sur les signatures sont vulnérables à plusieurs failles. Premièrement, ils ne détectent pas les nouvelles attaques, ce qui peut être très dommageable, surtout dans le cas d'un ver qui se propage très rapidement.

De plus, il est possible de fragmenter les paquets qui sont envoyés pour faire en sorte que le système de détection d'intrusion ne trouve jamais la signature d'attaque qu'il cherche parce que celle-ci est dispersée dans plusieurs paquets (Ptacek et Newsham, 1998). Il existe aussi des outils qui introduisent du polymorphisme au moment de la création des attaques (Zovi, 2001; Moore, 2003), rendant la détection beaucoup plus difficile.

Pour être en mesure de détecter des attaques qui n'ont pas encore été observées, il est possible d'utiliser un *système de détection d'anomalies*. Ce type de système analyse le trafic sur un réseau et construit un modèle du comportement "normal" des acteurs. Quand un comportement anormal est observé, le système déduit qu'une intrusion est en cours et envoie une alerte à l'administrateur.

Ce type de système est connu pour générer un grand nombre de faux positifs. Tous les comportements qui n'ont jamais été observés par le système de détection d'intrusion sont marqués comme étant anormaux alors qu'ils sont, la plupart du temps, des transactions légitimes mais moins habituelles.

Le positionnement des systèmes de détection d'intrus est aussi un facteur clé dans la qualité de ces solutions. Les *systèmes basés sur l'hôte* sont installés sur les stations de travail et analysent les divers journaux système à la recherche d'information sur une intrusion éventuelle. Afin de centraliser la détection, les *systèmes basés sur le réseau* écoutent les transactions réseaux à un endroit stratégique de celui-ci pour détecter les comportements malicieux.

2.7.3 Télescopes réseau

Un télescope réseau est un ensemble d'outils utilisés pour récolter des statistiques concernant le trafic dirigé vers une portion inutilisée de l'Internet [1]. Vu qu'aucun trafic légitime n'est dirigé vers ces adresses, le trafic récolté provient la plupart du temps d'agents malicieux ou de systèmes mal configurés. L'analyse du trafic qui se rend au télescope réseau donne l'opportunité aux chercheurs de mieux comprendre des phénomènes de sécurité informatique comme les attaques par déni de service, les balayages de réseau et la propagation des vers informatiques (Moore *et al.*, 2003b).

La tâche de collecte d'informations et de statistiques sur le trafic de l'Internet est

[1] Certains chercheurs ont aussi utilisé ce terme pour décrire l'écoute passive de trafic sur un très grand segment utilisé de réseau. Notre définition de télescope de réseau exclut les segments utilisés de l'Internet.

très difficile. Pour obtenir des mesures exactes, il faut être en mesure d'écouter du trafic à plusieurs points du réseau, amassant ainsi une quantité immense de données. L'utilisation des téléscopes réseau permet de cibler le trafic anormal. On peut ainsi amasser une quantité traitable de trafic.

Les téléscopes réseau ont été utilisés pour étudier les modèles de propagation des vers lors de grandes épidémies. Par exemple, grâce à leurs télescopes réseau, des chercheurs du San Diego's Super Computing Center ont pu compiler des statistiques sur la propagation du ver CodeRed en 2001 (Moore *et al.*, 2002). Pang *et al.* (2004) utilisent un télescope réseau pour caractériser le trafic anormal qui est sans cesse propagé dans l'Internet.

Le trafic dirigé ver un télescope réseau peut aussi être utilisé pour étudier les attaques par déni de service. Par exemple, en analysant les messages d'erreur ICMP et les paquets TCP de type SYN-ACK ou RST, on peut déduire qu'un attaquant effectue une attaque en utilisant des fausses adresses IPs. On peut aussi étudier les paquets TCP de type SYN qui sont dirigés vers le télescope. Ces types de paquets dénotent souvent une activité de balayage qui est en cours, soit par un attaquant, un outil automatisé ou un maliciel.

Quand un télescope de réseau observe une augmentation significative d'un certain type de requêtes, on peut déduire qu'un ver est en action sur l'Internet. Par contre, cette technique de détection des vers ne pourra peut pas être utilisée pour détecter les vers qui utilisent une liste pré-définie de victimes puisque ceux-ci n'effectuent pas de balayage réseau pour identifier leurs cibles.

2.7.4 Pots de miel

Un pot de miel, *honeypot* en anglais, est un ordinateur qui est configuré pour se faire attaquer et compromettre par des pirates informatiques [2] . C'est un serveur non sécurisé qu'on place sur l'Internet encadré de composants de surveillance pour enregistrer toutes les attaques qu'il subira. L'avantage d'un pot de miel est qu'il fournit de l'information sur les activités des pirates malveillants. Étant donné que le seul but d'un honeypot est de se faire attaquer, on peut considérer que toutes les requêtes qui sont faites vers ce serveur sont malveillantes.

Le projet Honeynet (réseau de pots de miels) a été lancé par une trentaine de

[2]Ces systèmes sont aussi parfois appelés *leurres informatiques*

chercheurs américains avec l'intention de conduire des recherches sur les techniques, outils, tactiques et motifs des pirates informatiques (Curran *et al.*, 2005; The Honeynet Project, 2002).

Les pots de miel sont souvent utilisés dans le cadre de recherches sur les logiciels malicieux. Par exemple, les chercheurs d'IBM ont créé un système de détection de vers (Riordan *et al.*, 2005) qui feint de répondre aux requêtes faites sur des adresses IPs qui ne sont pas allouées. Un cadre de création d'anti-vers a aussi été développé à l'aide de pots de miel pour capturer les vers informatiques (voir la section 2.7.5).

Les pots de miel sont utilisés pour amasser des données sur l'utilisation des réseaux de zombies (*botnets*) sur Internet (McCarty, 2003). Ces zombies sont utilisés pour espionner les utilisateurs d'ordinateurs et coordonner des attaques par déni de service distribuées (Holz, 2005). À l'aide d'un pot de miel qui est infecté, on peut voir toutes les commandes que le pirate envoie à ses esclaves et mieux comprendre l'utilisation de ces réseaux de zombies.

2.7.5 Génération d'anti-vers

Considérant que les vers se propagent très rapidement, il est nécessaire d'implanter des défenses automatisées pour s'assurer d'avoir le temps de réagir avant que la majorité des hôtes d'un réseau soient infectés. C'est dans le but d'organiser une défense rapide que la création de vers bénéfiques ou d'anti-vers a été proposée (Castañeda *et al.*, 2004).

Il est possible de créer automatiquement un ver qui utilisera les mêmes moyens de découverte de cible et d'infection qu'un ver malicieux. Les objectifs d'un anti-ver sont les suivants :

- Arrêter les dommages causés par un ver malicieux,
- Arrêter la propagation et l'infection du ver malicieux,
- Réparer les dommages causés par le ver malicieux
- Désinfecter les autres hôtes qui sont susceptibles d'être infectés.

Pour pouvoir atteindre ces objectifs, l'anti-ver doit se propager plus rapidement que le ver malicieux original. Par contre, il doit aussi minimiser la quantité de trafic de réseau qu'il génère pour ne pas faire plus de dégâts que le ver original.

Le concept d'anti-ver a déjà été observé sur l'Internet. Par exemple, le ver *Welchia* qui s'est propagé sur l'Internet avait comme objectif d'appliquer les rustines de mise

à jour aux systèmes qui avaient été infectés par le ver *Blaster*. *Welchia* se propageait six fois plus rapidement que *Blaster*, créant ainsi une surcharge des infrastructure de réseautique. Les vers *CRClean* et *CodeGreen* ont aussi été développés pour désinfecter les systèmes ayant été infectés par le ver *CodeRed*. Finalement, le ver *Cheese* a été conçu pour identifier et désinfecter les systèmes ayant été infectés par le ver *Lion*.

Pour qu'un anti-ver soit efficace, il doit être lancé sur le réseau le plus rapidement possible après l'apparition d'un ver malicieux. Sa conception doit donc être automatisée. Pour se faire, un cadre doit être créé pour générer automatiquement un anti-ver. Ce cadre fonctionne en trois étapes :

1. **Détection et capture du ver malicieux.** À l'aide d'un pot de miel, on détecte l'infection et on compare l'image du système infecté avec une image de sauvegarde qui n'est pas infectée.
2. **Analyse du ver.** Identification des divers composantes du ver. Principalement, on cherche à isoler la composante qui effectue la découverte de nouvelles cibles et la charge active malicieuse.
3. **Génération de l'anti-ver**. On garde toutes les composantes du ver malicieux, incluant le vecteur d'attaque. On modifie la charge active pour qu'elle effectue les tâches de désinfection au lieu de l'infection. Pour se faire, on insère un message dans le ver et on le fait attaquer une machine virtuelle isolée. Si le nouveau ver a le comportement attendu, on peut le lancer sur le réseau.

Un anti-ver peut utiliser quatre modèles de propagation :

- **Passif.** Le ver s'installe sur un serveur et attend de recevoir des messages d'attaque d'un ver malicieux pour ensuite essayer de se propager vers la machine infectée et la désinfecter. Cette technique a l'avantage de ne pas générer beaucoup de trafic réseau mais elle ralentit la vitesse de réparation.
- **Actif.** Le ver utilise une technique active de découverte de nouvelles cibles. La technique la plus répandue est de générer des adresses IP aléatoirement et d'essayer de s'y propager. Cette technique est efficace mais génère beaucoup de trafic réseau pouvant entraver l'opération normale des installation réseau.
- **Hybride.** Le ver se propage de façon active pendant un certain temps avant de se propager passivement.
- **Systèmes de détection d'intrusion.** Les systèmes de détection d'intrusion en place sont utilisés pour identifier les systèmes sur le réseau qui sont infectés,

l'anti-ver se propage seulement vers ces hôtes.

L'utilisation d'anti-vers peut sembler intéressante pour la défense des réseaux informatiques mais plusieurs points freinent l'adoption de cette technique. Premièrement, les conséquences légales de l'intrusion automatisée pour désinfecter un système peuvent être graves. De plus, l'anti-ver doit se propager plus rapidement que le ver malicieux mais aussi ne pas déranger les infrastructures réseautiques, ce qui est presque impossible.

2.8 Avenues de recherche

Dans ce chapitre, nous avons défini les différents types de maliciels qui ont été observés sur l'Internet. Nous avons ensuite décrit les plus grandes épidémies observées à ce jour dont l'épidémie du ver *Slammer* qui infecta 90% des systèmes vulnérables en moins de dix minutes et CodeRed qui infecta presque 360 000 systèmes en une demi-journée. En observant les épidémies "historiques", on constate qu'aucun des maliciels qui ont été lancés sur l'Internet n'était optimal. En rétrospective, chacun de ces maliciels aurait pu être amélioré d'une façon ou d'une autre pour causer plus de dégâts. Ces observations nous mènent à la conclusion que le pire reste encore à venir en terme d'attaques de maliciels.

Pour mieux comprendre la menace posée par les maliciels, nous avons étudié différentes taxonomies qui séparent le comportement des maliciels en différentes catégories. Certains chercheurs ont essayés de comprendre la dynamique des attaques de maliciels. Nous avons présenté les deux principales approches utilisées soit la modélisation et la simulation. Finalement, nous avons survolé les évolutions possibles des maliciels proposées par plusieurs chercheurs.

Pour comprendre les changements que subiront les maliciels dans les prochaines années, il est très important de bien comprendre leur fonctionnement. Les approches par simulation et modélisation présentées jusqu'à ce jour ont encore beaucoup de points à améliorer. Le plus gros reproche qu'on peut faire à ces deux techniques est le manque d'exactitude. Il est très difficile de créer des modèles ou des simulateurs de propagation de maliciels qui tiendront compte de tous les facteurs qui influencent la propagation.

Afin de mieux comprendre l'impact des différents facteurs qui influencent la performance des maliciels, nous devons définir un cadre clair pour les étudier et utiliser

une approche d'émulation pour observer leur comportement dans un environnement réseau réel. C'est ce que nous ferons dans les chapitres suivants.

CHAPITRE 3

Modèle structurant de la performance des maliciels

3.1 Analyse de performance

Afin d'évaluer les performances des maliciels, on se doit de formaliser un cadre d'analyse. Ce cadre d'analyse permettra de décrire les logiciels malicieux, les paramètres qui définissent leurs performances ainsi que l'environnement dans lequel ils évoluent. La figure 3.1 donne un aperçu de l'interaction entre les indices de performances, les caractéristiques des maliciels et leur environnement.

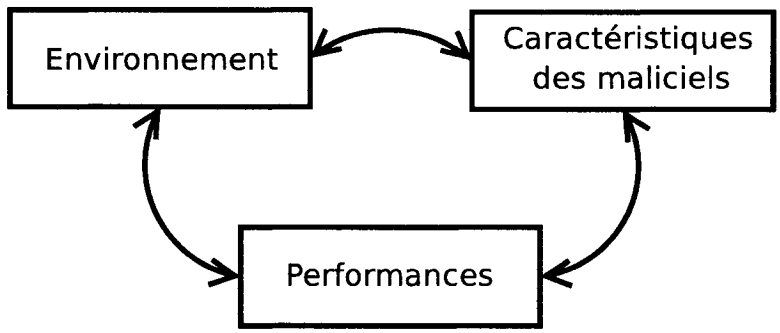

FIGURE 3.1 Modèle structurant

Cette figure montre que les trois composantes principales de l'évaluation de performance sont intimement liées et ont une influence réciproque. L'environnement englobe tous les facteurs qui ne sont pas contrôlés par le programmeur de maliciels ; Par exemple, la topologie d'un réseau qu'il veut attaquer, les systèmes de défense mis en place ainsi que les versions de logiciels utilisées. Les caractéristiques des maliciels sont, à l'inverse de l'environnement, les facteurs que le programmeur contrôle. Cette catégorie de facteurs inclut les choix de programmation pour la découverte de victimes, la propagation, la charge active et tous les mécanismes de défense du maliciel

qui peuvent être implantés. Finalement, la catégorie "performance" inclut tous les facteurs de performance qui peuvent être évalués pour déterminer si un maliciel est performant, c'est à dire s'il atteint d'une manière efficace ses objectifs.

Ce cadre à trois facteurs nous permet de formuler des observations sur les performances des logiciels malicieux. L'équation suivante décrit la situation où un programmeur de maliciels veut évaluer les performances de certaines caractéristiques de son programme. Il utilise un environnement (E) fixe et évalue des indices de performance (P) fixes en faisant varier les caractéristiques (C_1 et C_2) pour comparer leurs impacts sur l'indice de performance P.

$$(E \text{ fixe}, P \text{ fixe} \mid P(C_1) \stackrel{?}{<} P(C_2)) \tag{3.1}$$

La variation d'un indice de performance en fonction du changement de l'environnement (E_1 et E_2), lorsque les caractéristiques et les indices de performance ne changent pas est décrite par comme suit :

$$(C \text{ fixe}, P \text{ fixe} \mid P(E_1) \stackrel{?}{<} P(E_2)) \tag{3.2}$$

Finalement, les deux équations suivantes décrivent l'interaction entre les indices de performance et l'environnement d'opération du maliciel :

$$(C \text{ fixe} \mid P_1(E) \stackrel{?}{=} f(P_2(E))) \tag{3.3}$$

$$(E \text{ fixe} \mid P_1(C) \stackrel{?}{=} f(P_2(C))) \tag{3.4}$$

Ce type de relation aide à exprimer les compromis qu'un attaquant devra faire entre deux critères de performance distincts P_1 et P_2 (potentiellement reliés à deux objectifs distincts) et les caractéristiques de l'environnement qu'il veut attaquer.

3.2 Objectifs des maliciels

Nous basons notre analyse de performance des logiciels malicieux sur la supposition que ces programmes informatiques sont créés pour atteindre un objectif. Nous présentons ici les motivations qui peuvent amener un programmeur à créer des maliciels ainsi que les critères de performance associés à ces objectifs. Cette liste d'objectifs n'est pas exhaustive mais nous croyons qu'elle représente bien la situation actuelle.

Ces motivations risquent de se diversifier dans le futur avec l'adoption généralisée des technologies de l'information et de télécommunication.

3.2.1 Fraude

L'une des utilisations les plus fréquentes des maliciels est la fraude. Cet objectif, qui utilise la déception et de fausses informations pour augmenter les gains personnels en argent ou en propriété de l'attaquant, gagne en popularité. Les fraudes informatiques incluent le vol d'identité et le vol de numéros de carte de crédit. De nouvelles formes de fraude apparaissent avec le paiement par des compagnies de marketing pour l'installation de logiciels espions. Un attaquant peut gagner jusqu'à un dollar pour installer un logiciel espion sur un ordinateur, ce qui lui assure un revenu intéressant s'il réussit à contrôler un certain nombre de systèmes à l'aide de logiciels malicieux.

L'objectif principal de la fraude en ligne est d'accumuler des *montants* d'argent. Ces sommes peuvent donc être un bon indice de performance. Aussi, la *crédibilité* des maliciels tentant d'effectuer la fraude déterminera le taux de succès de ceux-ci. Plus un logiciel est crédible, plus les utilisateurs lui feront confiance et accepterons de saisir leurs informations personnelles ou de l'exécuter sans se méfier. Considérant que la fraude est une activité illégale, l'auteur de maliciel devrait aussi rester *anonyme*. Finalement, le *nombre* de systèmes infectés augmente les sommes d'argent potentiellement détournées.

3.2.2 Vol d'information

L'objectif du vol d'information est de pénétrer un réseau et d'y voler des données sensibles. Ces informations peuvent être des plans, un code source, des bilans financiers ou des communications confidentielles. Ces informations peuvent ensuite être utilisées par l'attaquant pour faire du chantage aux victimes ou être revendues à un tiers parti. Malgré que ce type d'attaque n'ait pas souvent été observé sur l'Internet, il pose un très grand danger, surtout pour les grandes compagnies et les agences gouvernementales car il s'agit d'un des outils les plus utilisés pour l'espionnage industriel.

Dans le cadre d'un vol d'information, l'attaquant veut s'introduire le plus profondément possible dans un réseau pour trouver des informations sensibles. Le plus grand taux de *pénétration* indiquera donc une meilleure performance. Le taux de pénétration peut se calculer comme le nombre de systèmes pouvant être compromis

par le logiciel malicieux par rapport au nombre total de systèmes dans l'environnement. La *rapidité* est aussi un facteur à considérer. La vitesse de propagation d'un maliciel peut être calculée en fonction du nombre de systèmes infectés par minute. Encore une fois, l'*anonymat* de l'auteur devra être conservé à tout prix. La *persistance* et la *furtivité* d'un agent malicieux sur les systèmes infectés augmentent les performances dans ce cas puisque plus l'agent peut s'exécuter longtemps, plus il a de chances d'accéder à des informations importantes. Pour augmenter les probabilités d'accéder à des informations sensibles, la *localisation* de la cible infectée doit être bien choisie. Par exemple, un maliciel s'exécutant sur un serveur de courriels aura accès à plus d'informations que sur un poste de travail.

3.2.3 Vente d'accès (botnets)

Un nouvelle mode dans l'utilisation des logiciels malicieux est de les joindre en réseaux de zombies (*botnets*). Ces réseaux de systèmes infectés peuvent être utilisés pour conduire des attaques par déni de service, chercher des informations sur les disques des victimes ou envoyer des courriels non sollicités (pourriels). On observe une tendance chez les "propriétaires" de botnets à louer leurs réseaux à d'autres personnes mal intentionnées. La location de ressources comme un *botnet* est un objectif de plus en plus observé sur l'Internet. Une attaquant infecte un grand nombre de systèmes et en loue ensuite l'accès en échange d'argent ou d'autres ressources.

Pour avoir de la valeur, l'accès aux systèmes compromis doit être contrôlé pour seulement permettre au plus offrant d'utiliser ces ressources. Des mécanismes d'*authentification* doivent donc être présents pour assurer la performance de ce type de maliciel. Le *nombre* de noeuds infectés sera aussi un indice de performance. En plus du nombre de systèmes infectés, les performances pour les dénis de service ou la cueillette d'information augmenteront avec la *bande passante* en amont disponible. On peut calculer approximativement la bande passante du réseau en additionnant la bande passante disponible pour chacune des stations qui composent le réseau.

3.2.4 Destruction

Une utilisation très simple mais malheureusement très efficace des maliciels est la destruction. Les dégâts peuvent être causés soit par la suppression de fichiers importants sur les systèmes, soit par la saturation des liens des réseaux ou par le bris

de composantes physiques. La destruction n'est pas seulement causée par la charge active du maliciel. Plusieurs maliciels ont causé des dégâts importants seulement avec leur trafic de découverte de nouvelles cibles et de propagation. Par exemple, le ver *Slammer* (voir la section 2.2.4) a rendu de grandes parties de l'Internet non accessibles et causé des changements importants dans les tables de routage sans toutefois avoir de charge active. Des bris matériels peuvent être causés par les maliciels à cause des nouvelles composantes électroniques qui sont maintenant configurables par des logiciels. Par exemple, un maliciel qui réussirait à changer le mot de passe du BIOS d'un ordinateur ou à diminuer la vitesse du ventilateur de refroidissement causerait des bris matériels importants.

Les indices qui permettront d'évaluer la performance d'un maliciel visant à effectuer le plus de dommage possible dans un réseau sont le *nombre* d'hôtes infectés, la *rapidité* de propagation, la *bande passante* disponible et la *localisation* des systèmes infectés. Le nombre d'hôtes infectés et la bande passante disponible sont deux facteurs qui permettent de déterminer la quantité de dégâts qui pourront être effectués lors d'attaques de dénis de service réseau. Le nombre de systèmes infectés et leur localisation peut aussi amplifier la quantité de dégâts physiques qui peuvent être engendrés en reconfigurant des composantes matérielles. La rapidité de propagation peut elle aussi effectuer des dénis de service sur le réseau en surchargeant les liens avec du trafic de recherche de victimes et de propagation.

3.2.5 Opérations d'informations

Comme discuté dans le guide *Opération d'informations* produit par les Forces canadiennes (2005), une opération d'information vise à assurer la supériorité du commandement par rapport à celui de l'ennemi. Les opérations d'informations peuvent améliorer la visualisation du champs de bataille, la maîtrise du rythme d'opération et la synchronisation des unités. Ce type d'opération peut viser l'acquisition d'informations auprès de l'ennemi (service de renseignement), la propagation d'informations erronées et la désorganisation des forces ennemies.

Les performances d'un logiciel malicieux effectuant des opérations d'informations peuvent être évaluées par sa *rapidité* d'action, la *quantité d'informations* qu'il réussit à acquérir, la *localisation* des noeuds infectés ainsi que l'*exposition* envers le public cible de la désinformation qu'il effectue.

3.2.6 Gloire et notoriété

La plus vieille motivation pour créer des logiciels malicieux est, sans contredit, la recherche de reconnaissance par ses pairs (*Fame and Glory*). Les premiers maliciels étaient des preuves de concepts créés pour montrer qu'un programmeur avait les compétences pour programmer un logiciel qui se copie de façon indépendante. Dans le cas de la recherche de gloire, le programmeur de logiciel malicieux cherche à avoir le plus grand impact possible sur l'Internet et plus particulièrement dans les médias.

Les indices pour évaluer les performances d'un maliciel ayant pour but de faire connaître son auteur sont donc l'*originalité*, l'*exposition* au public cible, la *localisation* des systèmes infectés et la quantité de *dommages* infligés. L'originalité est un facteur très important pour que le programmeur soit reconnu par ses pairs comme étant plus compétent que la moyenne car elle prouve que celui-ci n'a pas copié l'idée d'une autre personne en créant son logiciel. De la même manière, la localisation des systèmes infectés ainsi que la couverture générale de l'Internet sont importants pour avoir le plus d'impact possible sur les utilisateurs et sur les médias. Finalement, un maliciel qui inflige une grande quantité de dégâts matériels ou numériques a plus de chances de recevoir une couverture médiatique importante et donc d'apporter de la notoriété à son auteur.

3.3 Critères de performance

Une fois élaborée la liste des objectifs qui peuvent pousser un utilisateur malicieux à créer un maliciel, on est en mesure d'établir la liste des critères de performances qui s'appliquent pour chacune de ces catégorie d'objectifs. Le table 3.1 montre un résumé des différents critères de performance qui sont à considérer en fonction de l'objectif d'un maliciel.

TABLEAU 3.1 Critères de performance des maliciels en fonction des objectifs

Fraude	Vol	Location	Destruction	Op. d'info	Gloire
Montants	Pénétration	Authentification	Nombre	Rapidité	Originalité
Crédibilité	Furtivité	Nombre	Rapidité	Exposition	Localisation
Anonymat	Persistance	Bande passante	Bande passante	Localisation	Exposition
Nombre	Localisation		Localisation	Information	Dommages
	Anonymat		Dommages		

Ces critères de performance sont très variés d'un point de vue scientifique. Plusieurs d'entre eux sont facilement mesurables et représentent des données intéressantes qui peuvent être récoltées lors d'expériences en laboratoire ou même directement sur l'Internet. D'un autre côté, certains indices de performance sont très difficiles à évaluer de manière objective. Le tableau 3.2 montre l'organisation des critères de performance en fonction de leur évaluation.

TABLEAU 3.2 Facilité d'évaluation des critères de performance

Facilement mesurables	Possiblement mesurables	Difficilement mesurables
Couverture	Furtivité	Argent amassé
Nombre de systèmes infectés	Pénétration	Crédibilité
Rapidité de propagation	Anonymat	Informations volées
Situation des systèmes infectés	Sécurité	Originalité
Bande passante	Persistance	Désinformation
	Dommages / Disruption	

Comme présenté à la section 3.1, notre modèle d'analyse des performances de maliciels se base sur trois composantes : les indices de performance, l'environnement et les caractéristiques. Cette section a développé une liste d'indices de performance en portant attention à la faisabilité de leur observation. Les prochaines sections permettront de développer les deux autres aspects de l'évaluation des performances des maliciels.

3.4 Caractéristiques du maliciel

Les caractéristiques des logiciels malicieux sont tous les facteurs que le programmeur contrôle quand il crée un maliciel. Ce sont les choix de programmation que le programmeur fait et la valeur associée aux paramètres de ces caractéristiques qui auront une influence sur les performances des maliciels.

Pour évaluer les performances des logiciels malicieux, on doit établir une liste de caractéristiques qu'on pourra observer et évaluer leur impact sur les indices de performance (voir la section 3.2). Malgré le fait que plusieurs chercheurs aient avancé des taxonomies pour les logiciels malicieux (voir la section 2.3), nous présentons ici un nouveau point de vue pour les caractéristiques de logiciels malicieux. Pour ce, nous utiliserons le modèle OODA de caractérisation du commandement et contrôle.

3.4.1 Boucle OODA

Le concept de la *boucle OODA* a été développé par le colonel John Boyd de l'armée de l'air américaine suite à la guerre du Vietnam. (Boyd, 1987). Boyd est un ancien pilote d'avions de combat devenu stratège pour l'armée américaine. Selon Boyd, le secret de la victoire est la rapidité avec laquelle on prend des décisions, ce qui permet d'être plus rapide que l'adversaire.

Pour modéliser le comportement des maliciels et classifier leurs caractéristiques, nous utilisons le modèle de la *boucle OODA*. Pour que notre modèle soit adapté, nous devons nous baser sur l'hypothèse que *les maliciels sont créés pour atteindre des objectifs.* Dans ce modèle, les maliciels sont vus comme des agents plus ou moins autonomes qui tentent d'atteindre leurs objectifs. L'atteinte des objectifs est calculée à l'aide des indices de performances.

La figure 3.2 montre la boucle telle que présentée par le colonel Boyd. Les quatre éléments clé de la boucle sont l'Observation, l'Orientation, la Décision et l'Action, d'où le nom du modèle.

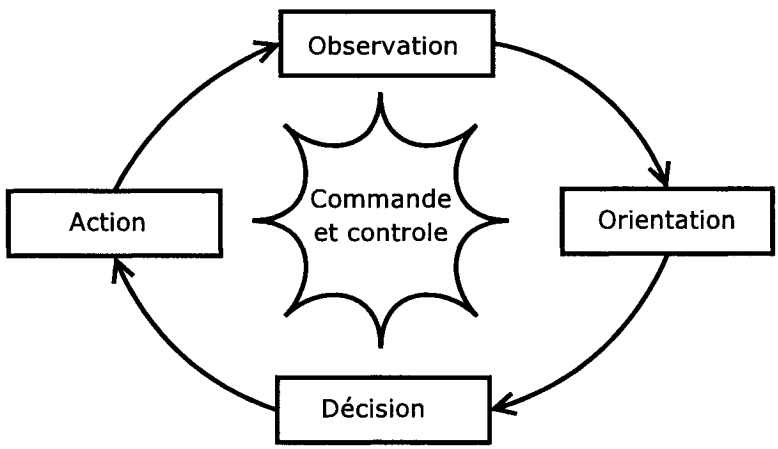

FIGURE 3.2 Boucle OODA

Les quatre phases de la boucle OODA représentent le cheminement d'une unité qui tente d'atteindre son objectifs :

1. **Observation.** Cette première phase tente de répondre à la question "Que vois-tu ?" Dans cette phase, l'agent récolte des informations brutes sur ce qui l'entoure et possiblement sur les changements qui se sont opérés depuis sa dernière

phase d'observation. L'observation est l'étape où l'agent s'informe sur l'environnement dans lequel il se situe.

2. **Orientation.** C'est dans cette phase que l'agent considère les connaissances qu'il a par rapport à lui-même, à son environnement, à ses actions passées et à ses alliés. Ces informations sont ajoutées aux observations faites dans la phase précédente et traitées pour générer un modèle cognitif qui sera utilisé pour prendre des décisions pertinentes.
3. **Décision.** Après avoir construit un modèle cognitif de la situation dans laquelle il se trouve, l'agent doit décider quelle action prendre. Cette décision doit être faite pour augmenter les chances d'atteindre les objectifs qui ont été déterminés au préalable ou pour effectuer une tâche qui lui a été commandée par un mécanisme de commande et contrôle. C'est aussi à cette étape que les instructions reçues par le commandement sont considérées.
4. **Action.** L'action est simplement l'implémentation de la décision prise pour se rapprocher d'un objectif. Cette phase inclut toutes les actions qu'un agent peut exécuter, par exemple de se propager à un autre système ou simplement d'arrêter d'agir pour un certain temps pour diminuer les chances d'être détecté.

3.4.2 Boucle de boucles OODA

Une des forces du modèle OODA est sa flexibilité. En effet, il est possible de l'utiliser pour représenter une chaîne de commandement ou l'action conjointe d'un groupe d'agents. Ceci est réalisé en imbriquant des boucles OODA. La figure 3.3 montre l'imbrication de boucles OODA pour décrire une forme générale de commandement et contrôle.

L'imbrication de boucles OODA permet de modéliser à la fois les comportements tactiques et stratégiques. Les comportements tactiques sont les actions de bas niveau prises par les unités déployées sur le terrain et représentés par les boucles "internes". Dans notre cas, les actions tactiques sont les vecteurs d'attaque et de défense qui peuvent être utilisés par les maliciels. Sur le plan stratégique, on trouve les preneurs de décision qui envoient des ordres pour atteindre des objectifs plus généraux. Les actions sur le plan stratégique sont donc de donner des ordres aux unités et le résultat de ces ordres revient dans la phase d'orientation de la boucle "externe". Pour les maliciels, l'instance stratégique est souvent un attaquant qui utilise des maliciels comme

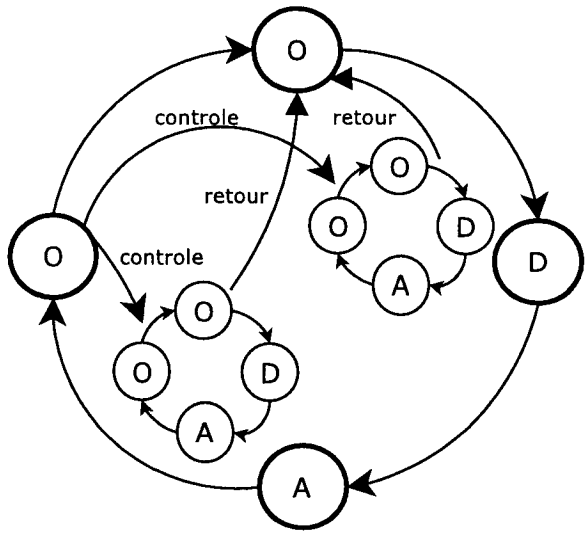

FIGURE 3.3 La boucle externe représente la prise d'actions stratégiques tandis que les boucles internes représentent le comportement des unités déployées sur le terrain.

outils pour atteindre un objectif général (voir la section 3.2 pour une description des objectifs généraux).

3.4.3 Application du modèle OODA au cas du ver Slammer

Pour montrer que le modèle de la boucle OODA s'applique bien aux logiciels malicieux, nous avons divisé les caractéristiques d'un des ver informatiques les plus rapide de la courte histoire de l'Internet, le ver Slammer. Comme nous l'avons décrit à la section 2.2.4, le ver Slammer a infecté près de 90% des victimes potentielles en dix minutes. Ce ver exploitait une faille de sécurité dans le logiciel SQL server de Microsoft à l'aide d'un simple paquet de données envoyé avec le protocole UDP.

Selon le modèle OODA, le ver Slammer n'a pas vraiment de caractéristiques d'observation puisque la seule opération qu'il effectue pour trouver une nouvelle victime est de générer aléatoirement une adresse IP. Vu que le ver n'effectue aucun traitement sur l'environnement qui l'entoure, ses alliés ou victimes potentielles, on considère qu'il ne possède aucune caractéristique d'orientation. Le fait de ne pas avoir de phase d'orientation a pu nuire au ver Slammer puisqu'il s'est propagé très rapidement et que cette propagation a complètement congestionné plusieurs liens sur l'Internet et donc ultérieurement freiné sa propagation et accéléré sa détection et son éradication. Une orientation pour considérer la vitesse optimale de propagation aurait probable-

ment permis d'améliorer les performances du ver Slammer. La phase de décision du ver Slammer est très simple : il attaque toutes les adresses générées et se propage du même coup vers toutes celles qui sont vulnérables. Slammer n'a pas de charge active, c'est à dire que la seule action qu'il effectue après avoir infecté un système est de continuer sa propagation. On peut donc dire que ce ver a une seule caractéristique d'action et c'est la propagation vers d'autres victimes potentielles.

3.5 Environnement

Dans notre étude des maliciels, l'environnement inclut toutes les caractéristiques qui influencent les performances de ceux-ci et qui ne sont pas contrôlées par le programmeur malicieux. Ce sont ces caractéristiques de l'environnement qui représentent la plus grande difficulté à surmonter pour un attaquant et, heureusement, la meilleure ligne de défense pour un administrateur système. Même si les caractéristiques de l'environnement ne sont pas contrôlables par un attaquant, les administrateurs du réseau peuvent souvent les modifier et ainsi diminuer les performances d'un attaque de logiciels malicieux.

Le plus souvent, les maliciels sont influencés par les caractéristiques suivante de leur environnement :

- **Topologie des réseaux.** Division du réseau en plusieurs sous réseaux pour des raisons de performance ou de sécurité. Par exemple, on sépare souvent les serveurs de production exposés à l'Internet pour les placer dans une zone démilitarisée (DMZ). La topologie des réseau implique qu'un maliciel ne peut pas accéder à tous les noeuds d'un réseau à partir de tous les sous réseaux, il doit se trouver dans une zone proche, ce qui rend la tâche d'identification de nouvelles victimes et de propagation plus difficile.
- **Systèmes d'exploitation utilisés.** La plupart des maliciels se propagent en exploitant une faille de sécurité dans un logiciel installé sur un système d'exploitation. Très peu de vecteurs d'attaque sont identiques quand les systèmes d'exploitation varient. La plupart des maliciels s'attaquent donc spécifiquement à un système d'exploitation. Si le maliciel se retrouve dans un environnement où aucun système n'est vulnérable à ses attaques, ses performances seront influencées négativement.
- **Services activés sur les systèmes.** Les services activés sur les systèmes

représentent les serveurs qui offrent des fonctionnalités aux utilisateurs, par exemple un serveur web ou un serveur de courriels. Les services qui sont activés peuvent contenir des failles de sécurité comme des débordement de tampons ou des chaînes de format. C'est par ces failles que les maliciels peuvent le plus souvent se propager, d'où l'importance de ce facteur dans l'analyse de l'environnement. Les maliciels peuvent détecter qu'un service est activé sur un serveur en tentant de s'y connecter via des protocoles réseau comme TCP ou UDP.

- **Équipement défensif déployé.** Les équipements qui peuvent être déployés pour améliorer la sécurité d'un réseau sont les murs pare-feu, les filtres de pourriels, les systèmes de détection d'intrusion et les systèmes de prévention d'intrusion. La présence ainsi que la position de ces équipements peuvent permettre de freiner une épidémie de maliciels.
- **Bande passante disponible.** La bande passante disponible aura un impact sur la propagation des maliciels puisque ces programmes doivent utiliser la bande passante pour identifier des nouvelles victimes et se propager, c'est à dire envoyer une copie de leur programme vers les victimes nouvellement infectées.
- **Virtualisation.** La virtualisation est utilisée dans les fermes de serveurs puisqu'elle permet un partage plus facile et efficace des ressources informatiques. La virtualisation permet aussi aux administrateurs réseau d'isoler plus facilement un système infecté par un logiciel malicieux et de le réparer rapidement. Cette technologie est aussi fréquemment utilisée pour faire la rétro ingénierie des maliciels, permettant d'étudier leur comportement dans un environnement contrôlé. La virtualisation a donc un impact sur la persistance et l'anonymat d'un maliciel.
- **Habitudes des utilisateurs.** Les habitudes d'utilisation des personnes sur un réseau ont un impact sur la propagation des logiciels malicieux. Si les utilisateurs sont vigilants et ouvrent seulement les fichiers qui proviennent de sources en qui ils ont confiance et s'ils gardent leurs systèmes à jour et sécurisés, ils rendent le terrain beaucoup plus difficile aux maliciels qui se propagent par la voie de fichiers infectés ou de failles de sécurité connues.

  En plus de l'éducation des utilisateurs, leurs habitudes d'utilisation du réseau ont un impact sur l'efficacité de détection et de prévention de comportements malicieux. En effet, il est beaucoup plus facile d'identifier un comportement anormal sur un réseau où tous les utilisateurs effectuent un nombre limité d'ac-

tions. Un réseau où tous les utilisateurs génèrent beaucoup de trafic varié créera beaucoup de faux positifs dans les systèmes de détection d'intrusion, ce qui augmente la chance qu'un maliciel ne soit pas détecté.

Pour l'instant, la plupart des maliciels utilisent une représentation simpliste de leur environnement ; en effet, ils se basent presqu'uniquement sur le fait qu'un port TCP ou UDP soit ouvert pour attaquer ou pas un système. Même la plupart des outils commerciaux de sécurité comme Core Impact et Canvas se basent sur l'ouverture de ports et les bannières retournées lors de la connexion aux ports ouverts. Une analyse plus profonde des habitudes des utilisateurs, de la bande passante et du positionnement des systèmes défensifs permettrait sûrement d'augmenter les performances d'un logiciel malicieux en créant un modèle cognitif plus riche favorisant une prise de décision plus complexe.

3.6 Utilisation du modèle de performance

En combinant le modèle de performance à trois facteurs, les indices de performances sélectionnés en fonction des objectifs du maliciel et la boucle OODA pour décrire les caractéristiques du maliciel, nous sommes en mesure d'évaluer efficacement les performances des maliciels.

Nous proposons d'utiliser notre approche pour créer un cadre de prototypage de logiciels de maliciels. Ces prototypes pourront être déployés dans un environnement de tests pour mieux comprendre les relations entre les caractéristiques, l'environnement et les indices de performances. Une meilleure connaissance de ces relations permet de préparer les infrastructures de défense de façon pro-active et de construire des réseaux qui résisteront aux attaques futures de maliciels.

Il est à noter qu'une grande partie de notre analyse portant sur les maliciels est aussi applicable aux systèmes de défense. Plusieurs solutions de détection et de prévention d'intrusion utilisent déjà des agents pour collecter des informations sur la sécurité du réseau et prendre action pour contrer les intrusions. La boucle OODA s'applique aussi à ces agents mais ce sont les objectifs qui changent. Les objectifs sont formulés en fonction des politiques de sécurité pour le réseau en place. Avec deux types d'agents indépendants qui peuvent évoluer dans des réseaux, il est maintenant possible d'envisager une coévolution de ceux-ci qui pourrait, à terme, produire des agents défensifs bien "préparés" et "entraînés" pour répondre aux menaces à venir.

Cependant, et dans le but plus immédiat de valider l'utilité de cette vision et de ces modèles de performance du maliciel, nous nous sommes proposés d'étudier un aspect plus concret : la relation entre la performance du maliciel en terme de propagation et le filtrage au niveau du réseau comme caractéristique de l'environnement. Ceci fait l'objet des chapitres suivants.

CHAPITRE 4

Étude théorique de la relation entre la performance et l'environnement

Dans ce chapitre, nous réalisons une étude théorique visant à approfondir notre compréhension de la relation qui existe entre le filtrage réseau et la vitesse de propagation des maliciels. Du point de vue du modèle à trois facteurs présenté au chapitre 3, l'étude décrite dans ce chapitre examine la relation entre une caractéristique de l'environnement et un indice de performance.

La section 4.1 donne une description générale de l'étude théorique. À la section 4.2, nous exposons un modèle mathématique que nous avons développé pour établir des prédictions sur la vitesse de propagation.

4.1 Propagation et filtrage réseau

La relation que nous nous proposons d'étudier est celle entre l'environnement dans lequel évolue un maliciel et un indice de performance. Nous posons la conjecture que le filtrage du trafic entre les sous-réseaux composant un réseau diminue la vitesse de propagation d'un maliciel. Cette conjecture est donc une instance de l'équation 3.3 présentée à la section 3.1 de ce mémoire.

$$(C \text{ fixe}, \mathcal{P} \text{ fixe} \mid P(E_1) \overset{?}{<} P(E_2)) \tag{4.1}$$

Dans l'instance particulière de l'équation de performance, nous considérons que l'indice de performance P est fixe. La performance du maliciel dans notre cas est calculée en fonction de la vitesse de propagation V_p. Cette vitesse de propagation est calculée en nombre de systèmes infectés pour une période de temps donnée. Les caractéristiques qui sont changées dans l'environnement sont le filtrage entre deux

sous-réseaux. Nous exprimons le filtrage comme étant le nombre de passerelles G ("gateway" en anglais), c'est à dire des systèmes qui sont connectés aux deux sous-réseaux observés. Si un système est connecté aux deux sous-réseaux, cela signifie qu'il peut communiquer avec tous les systèmes présents sur le réseau. Par contre, si un système est simplement dans un des sous-réseaux, il peut interagir uniquement avec les ordinateurs qui se trouvent dans le même sous-réseau. Les caractéristiques $C_{ob}, C_{or}, C_{de}, C_{ac}$ du maliciel que nous voulons modéliser sont fixes et décrites plus tard à l'aide de la boucle OODA. Si nous considérons V_p pour un intervalle de temps Δt fixe et les caractéristiques $C_{ob}, C_{or}, C_{de}, C_{ac}$ fixes, nous adaptons l'équation 3.3 pour arriver à l'équation suivante :

$$(G_1 < G_2 \mid V_p(G_1) \overset{?}{<} V_p(G_2)) \tag{4.2}$$

montrant que moins il y a de passerelles reliant deux sous-réseaux, plus la vitesse de propagation est lente.

Nous décrivons les caractéristiques fixes (C) du maliciel à l'aide de la boucle OODA :

- **Observation.** La phase d'observation est primitive et effectue un balayage aléatoire d'adresses pour déterminer où se situent les systèmes vulnérables aux attaques du maliciel.
- **Orientation.** La phase d'orientation du maliciel est inexistante. Nous aurions pu considérer une fonctionnalité d'orientation par communication entre les instances malicieuses mais nous avons décidé d'exclure cette option pour modéliser un maliciel plus près de la réalité actuelle.
- **Décision.** Les décisions prises par le maliciel étudié sont très simples et sont exprimées sous forme de règles. Ces règles dictent que si un système potentiellement vulnérable est trouvé, il est automatiquement attaqué. Ces décisions peuvent être décrites comme agressives parce que l'unique objectif est d'infecter le plus grand nombre de systèmes dans un temps le plus court possible.
- **Action.** Le maliciel peut effectuer deux actions différentes mais qui sont toujours faites l'une à la suite de l'autre. La première action est l'infection, c'est pendant cette phase que le maliciel infecte une nouvelle victime. La deuxième action est la propagation. Lors de cette phase, le code de l'agent malicieux est envoyé vers la victime nouvellement infectée pour que la victime devienne une

nouvelle instance malicieuse.

La caractéristique de l'environnement qui change entre les scénarios de modélisation est G, le nombre de systèmes interconnectant deux sous-réseaux. Nous considérons un réseau séparé en deux parties qui ne peuvent pas communiquer entre elles. Les seuls systèmes qui sont autorisés à établir des connexions vers tous les systèmes sont les systèmes qui sont connectés aux deux sous-réseaux. Par exemple, la figure 4.1 montre un réseau de où $G = 4$.

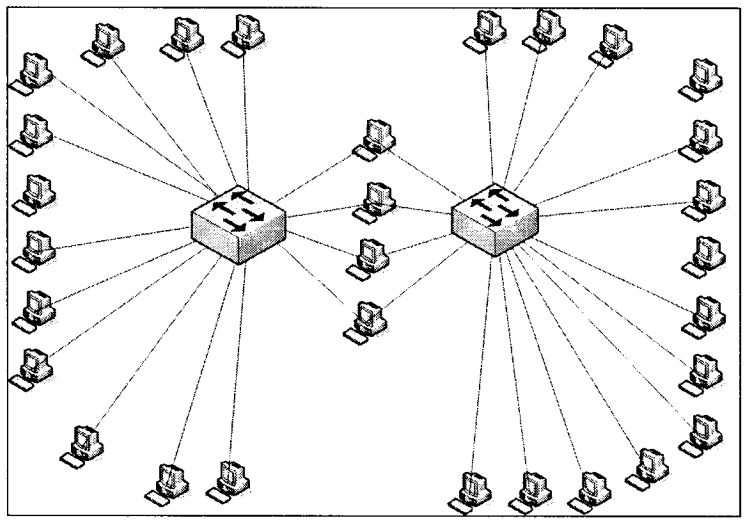

FIGURE 4.1 Topologie du réseau avec quatre systèmes interconnectés ($G = 4$)

Nous considérons également le cas où il existe seulement un réseau complètement interconnecté ; dans ce cas, $G = N$, où N est le nombre total de machines dans le réseau.

4.2 Modèle mathématique

Cette section décrit le modèle mathématique que nous avons utilisé pour modéliser le comportement d'une infection de maliciel dans un réseau où un certain niveau de filtrage est effectué.

4.2.1 Description

Pour décrire la vitesse de propagation d'un ver dans un réseau informatique, nous utilisons un processus de Markov. Le processus de Markov est un modèle stochastique

qui a la propriété d'être Markovien, c'est à dire que la probabilité de transition vers les états futurs dépend seulement de l'état présent. La probabilité de passer d'un état vers un autre dépend donc uniquement de l'état dans lequel se trouve le modèle au temps de la transition.

Dans notre cas, nous utilisons un processus de Markov pour représenter le nombre d'infections dans un réseau avec deux sous-réseaux. Les états sont identifiés par un vecteur à trois éléments, i, j et k. Ces trois éléments représentent respectivement le nombre de systèmes infectés dans le premier sous-réseau, le nombre de systèmes infectés qui interconnectent les deux sous-réseaux et le nombre de systèmes infectés dans le deuxième sous-réseau. L'état initial du système est toujours {1,0,0} parce que nous modélisons toujours une infection qui démarre avec un système infecté dans le premier sous-réseau de l'environnement. Nous définissons les ensembles I, J et K comme étant, respectivement, l'ensemble de machines dans le premier sous-réseau, l'ensemble de machines passerelles et l'ensemble de systèmes dans le deuxième sous-réseau. Ces ensembles incluent les systèmes infectés et ceux qui ne le sont pas. On peut donc calculer le nombre de systèmes qui ne sont pas infectés (vulnérables) par $|I| - i$.

Les transitions dans notre processus de Markov dépendent du nombre de systèmes qui sont infectés. Par exemple, si un système dans l'ensemble I est infecté, il est possible que lors de la prochaine infection, un autre système dans I soit infecté, soit un système dans J soit infecté. Le tableau 4.1 montre les différentes conditions nécessaires pour qu'une transition soit possible d'un état vers un autre.

TABLEAU 4.1 Transitions possibles dans le processus de Markov

Condition	Transition	Description
$i < \|I\|, i \geq 1$	$(i,j,k) \rightarrow (i+1,j,k)$	Infection de I vers I
$j < \|J\|, i \geq 1$	$(i,j,k) \rightarrow (i,j+1,k)$	Infection de I vers J
$i < \|I\|, j \geq 1$	$(i,j,k) \rightarrow (i+1,j,k)$	Infection de J vers I
$j < \|J\|, j \geq 1$	$(i,j,k) \rightarrow (i,j+1,k)$	Infection de J vers J
$k < \|K\|, j \geq 1$	$(i,j,k) \rightarrow (i,j,k+1)$	Infection de J vers K
$j < \|J\|, k \geq 1$	$(i,j,k) \rightarrow (i,j+1,k)$	Infection de K vers J
$k < \|K\|, k \geq 1$	$(i,j,k) \rightarrow (i,j,k+1)$	Infection de K vers K

Pour chaque transition du processus de Markov, nous voulons associer la probabilité qu'un système de l'ensemble X de départ infecte un système dans l'ensemble

d'arrivé Y. Nous définissons premièrement la probabilité qu'un système $\alpha \in X$ infecte un système β pour un réseau contenant A adresses.

$$\begin{aligned} \Pr(\alpha \text{ infecte } \beta) &= \sum_{\beta \in \text{ non infect.}} \Pr(\alpha \text{ choisit } \beta) && (4.3) \\ &= |\beta \text{ non infect.}| \cdot \Pr(\alpha \text{ choisit } \beta) && (4.4) \\ &= |\beta \text{ non infect.}| \cdot \frac{1}{A}. && (4.5) \end{aligned}$$

Par exemple, si on considère une infection de I vers I, où i représente le nombre de systèmes déjà infectés, alors la probabilité est calculée comme suit :

$$\Pr(\alpha \text{ infecte } \beta) = (|I| - i) \cdot \frac{1}{A}. \tag{4.6}$$

Soit x et y le nombre de systèmes infectés dans les ensembles de départ X et d'arrivée Y, respectivement, alors définissons la probabilité p d'infection par une seule machine β de Y par une machine α de X comme :

$$p := (|Y| - y) \cdot \frac{1}{A}, \tag{4.7}$$

tel que calculé à l'équation 4.5. Maintenant, notre modèle doit considérer le fait que plusieurs instances de maliciels tentent de se propager dans le même intervalle de temps. La probabilité qu'une machine quelconque de X infecte une machine quelconque de Y est alors :

$$\begin{aligned} \Pr(1 \text{ infection } X \rightarrow Y) &= \sum_{\alpha \in X} p \cdot \Pr(\text{aucune autre infection}) && (4.8) \\ &= x \cdot p \cdot \Pr(\text{aucune autre infection}) && (4.9) \\ &= x \cdot p \cdot (1-p)^{x-1} && (4.10) \end{aligned}$$

Pour simplifier l'analyse et la résolution de notre système, nous avons décidé de considérer que la transition d'un maliciel se fait à intervalles réguliers. Vu que les transitions sont régulières, nous effectuons la résolution du modèle en temps discret.

L'équation utilisée pour représenter la probabilité d'infection d'un sous-réseau vers un autre pourrait être améliorée pour tenir compte du fait que plusieurs transitions sont possibles dans un même intervalle de temps. Dans l'état actuel, l'équation

représente la probabilité d'une infection d'un sous-réseau vers un autre. L'équation pourrait, par exemple, exclure la portion Pr(aucune autre infection), ce qui augmenterait légèrement les probabilités de transitions. L'espérance du nombre de systèmes infectés augmenterait un peu plus rapidement, donnant un modèle plus pessimiste et probablement plus près de la réalité.

Si on considère, par exemple, la première ligne du tableau 4.1, la probabilité d'une infection de I vers I est

$$\Pr\left((i,j,k) \rightarrow (i+1,j,k)\right) = i \cdot p \cdot (1-p)^{i-1} \tag{4.11}$$

Dans notre modèle, un réseau ayant deux sous-réseaux avec chacun deux systèmes et un système assurant la connexion ($|I| = 2$, $|J| = 1$, $|K| = 2$) produirait le processus de Markov illustré à la figure 4.2

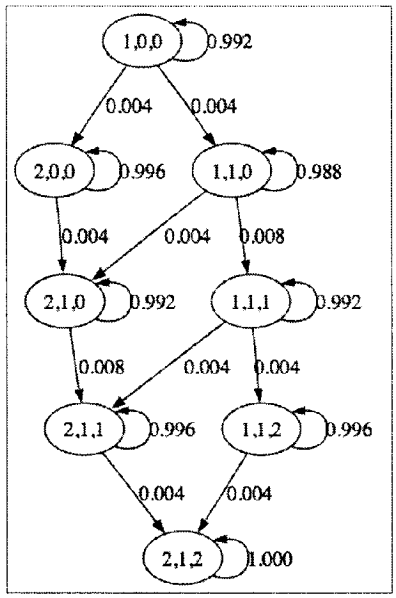

FIGURE 4.2 Processus de Markov pour $|I| = 2$, $|J| = 1$, $|K| = 2$ et $A = 256$

La figure 4.2 est un exemple très simple de modèle de Markov que nous pouvons générer. Il est à noter qu'aucune transition multiple n'est possible dans cet exemple. Une transition multiple est possible dans un réseau où deux machines ou plus réussissent à infecter plus d'un nouveau système dans un même intervalle de temps. Notre modèle ne tient pas compte de ces transitions multiples pour l'instant.

4.2.2 Résolution du modèle

Une fois le processus de Markov créé en fonction des conditions et des probabilités de transitions présentées à la section précédente, il est simple de générer la matrice de transition. Cette matrice de transition est composée des probabilités de transitions entre chacun des états du processus. Le tableau 4.2 montre la matrice de transition générée à partir de l'exemple de la figure 4.2. Cette matrice stochastique est bien une matrice de transition parce que la somme des probabilités de transition est égale à 1 pour chaque ligne. Les états sont représentés dans la matrice selon leur ordre lexicographique.

TABLEAU 4.2 Matrice de transition pour $|I| = 2$, $|J| = 1$, $|K| = 2$ et $A = 256$

	(1,0,0)	(2,0,0)	(1,1,0)	(2,1,0)	(1,1,1)	(2,1,1)	(1,1,2)	(2,1,2)
(1,0,0)	254/256	1/256	1/256	-	-	-	-	-
(2,0,0)	-	$65026/256^2$	-	$510/256^2$	-	-	-	-
(1,1,0)	-	-	253/256	1/256	2/256	-	-	-
(2,1,0)	-	-	-	254/256	-	2/256	-	-
(1,1,1)	-	-	-	-	254/256	1/256	1/256	-
(2,1,1)	-	-	-	-	-	254/256	-	1/256
(1,1,2)	-	-	-	-	-	-	254/256	1/256
(2,1,2)	-	-	-	-	-	-	-	1

L'état initial de notre modèle est toujours l'état où seulement un système dans i est infecté et tous les autres systèmes ne sont pas infectés. Nous multiplions la matrice de transition M par un vecteur dont tous les éléments valent 0, sauf le premier qui vaut un 1 (ce vecteur représente l'état initial du système). Si on considère la matrice de transition M et le vecteur p_0 le vecteur représentant le système à l'état initial, nous avons l'équation suivante :

$$\vec{p_1} = \vec{p_0} \cdot M. \tag{4.12}$$

De même, on peut calculer l'état du système au temps $t+1$ à l'aide de l'équation suivante :

$$\vec{p_{t+1}} = \vec{p_t} \cdot M. \tag{4.13}$$

La multiplication de la matrice de transition par le vecteur représentant l'état du système nous donne la probabilité que le système soit dans un certain état à chaque

itération. Dans le cadre de cette étude, nous prenons en considération que chaque itération du modèle théorique prend une seconde. Nous faisons cette approximation en sachant que le temps de connexion à un système prend une seconde dans le code du maliciel que nous tentons de modéliser.

À chaque itération de multiplication de la matrice de probabilité de transition M par le vecteur $\vec{p_t}$, nous pouvons multiplier le nombre de systèmes infectés par état par la probabilité que le système soit dans cet état pour obtenir l'espérance du nombre de systèmes infectés au temps t. Nous effectuons la multiplication du vecteur $\vec{p_t}$ par le nombre de systèmes infectés dans le processus de Markov à chaque itération pour savoir à chaque seconde l'espérance du nombre de systèmes infectés. Ce sont ces résultats qui sont présentés à la section 4.2.3.

4.2.3 Résultats sur l'évolution de l'infection

Nous avons effectué une résolution à l'aide de notre modèle théorique pour notre étude avec les trois niveaux suivants :

- Niveau 1 : aucun filtrage ($G = 30$).
- Niveau 2 : filtrage bas ($G = 4$).
- Niveau 3 : filtrage élevé ($G = 1$).

Le cas d'un réseau sans filtrage est représenté à l'aide de $G = |J| = 30$, $|J| = |K| = 0$ puisque nous considérons un réseau où tous les systèmes peuvent communiquer entre eux.

La résolution est en pratique la multiplication de la matrice de transition générée à partir du processus de Markov par un vecteur représentant l'état du système. Nous avons fait la génération de la matrice ainsi que les multiplications à l'aide d'un script programmé en Ruby. La sortie de ce script est l'espérance du nombre de systèmes infectés dans le réseau à chaque intervalle de temps (chaque seconde dans notre cas). La figure 4.3 montre les résultats obtenus avec les trois niveaux de filtrage.

Le tableau 4.3 montre les différents résultats numériques récoltés avec le modèle théorique. Vu que nous avons affaire à un modèle théorique qui comptabilise les probabilités d'être dans un certain état du processus de Markov, la probabilité d'avoir infecté la totalité des systèmes présents ne sera jamais égale a un. C'est pourquoi le résultat de la ligne représentant le temps pour infecter 30 systèmes est une approximation où l'espérance du nombre de systèmes infectés est de 29.90, représentant donc

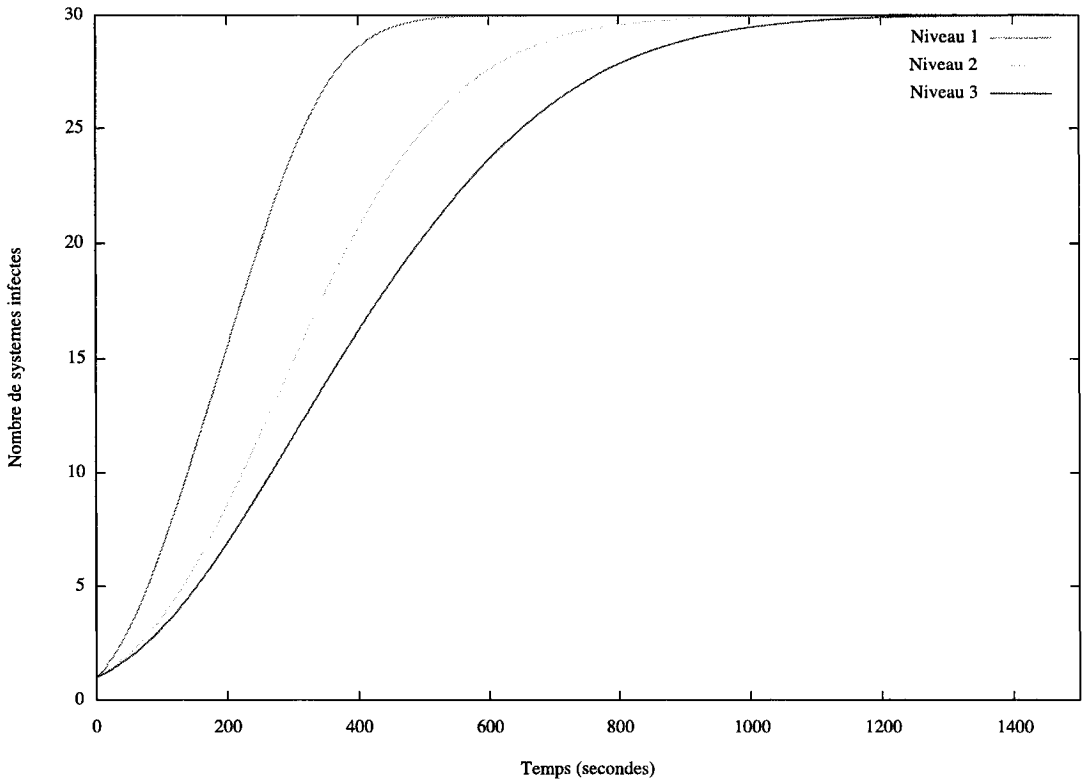

FIGURE 4.3 Comparaison des résultats théoriques

une probabilité de 99.7% que tous les systèmes soient infectés.

TABLEAU 4.3 Résultats numériques du modèle théorique

Temps pour infecter (s)	Niveau 1	Niveau 2	Niveau 3
10 systèmes	140	224	268
20 systèmes	252	388	492
25 systèmes	320	504	652
30 systèmes	532	944	1208

Ces résultats montrent ce que l'on peut déjà déduire intuitivement : plus il y a de filtrage appliqué entre les réseaux, moins la propagation de maliciels sera rapide. Cette diminution de la vitesse de propagation théorique est due au fait qu'il est plus difficile au maliciel d'infecter le deuxième sous-réseau puisqu'il est possible de l'atteindre par un nombre limité de connexions.

On constate que le temps pour infecter tous les systèmes passe de 532 secondes dans un réseau sans filtrage à 1208 pour deux sous-réseaux où seulement un système assure la connexion entre les sous-réseaux. Ces résultats sont très encourageants et montrent que le filtrage peut ralentir de 100% la vitesse de propagation d'un maliciel. La situation où seulement un système est connecté aux deux réseaux est par contre peu probable puisque peu fonctionnelle. L'exemple du scénario 2 montre que même si quatre systèmes sur 30 sont connectés aux deux réseaux, on augmente le délai avant l'infection du dernier système de 300 secondes, plus de cinq minutes. Nous voyons que le filtrage dans un réseau a beaucoup d'impact sur la performance des maliciels et même pour un nombre de systèmes infectés plus bas que trente, le temps d'infection de 20 et 25 systèmes sont aussi nettement plus long quand un filtrage est appliqué.

Dans le prochain chapitre, nous décrivons les expériences réalisées pour tenter de confirmer ces résultats théoriques.

CHAPITRE 5

Expérimentation sur la performance du maliciel

Pour mieux évaluer les performances des maliciels, nous avons créé un cadre de prototypage qui nous permettra de créer des prototypes. Ce chapitre décrit le processus de création d'un prototype de maliciel. La section 5.1 décrit l'idée générale de cet outil, la section 5.1.1 son architecture, la section 5.1.2 les fonctionnalités implantées et, finalement, la section 5.1.3 les mécanismes de sécurité utilisés pour effectuer les expériences.

Une fois les prototypes créés, ils ont été utilisés pour conduire des expériences visant à mieux comprendre les performances des maliciels, plus particulièrement la relation entre la vitesse de propagation et le filtrage réseau. La section 5.2 décrit le contexte technique de notre expérience et les résultats pratiques obtenus. Pour finir, la section 5.3 compare les résultats théoriques obtenus au chapitre précédent et les résultats pratiques.

5.1 Cadre de prototypage de maliciels MEF

Dans le contexte de ce mémoire, nous avons décidé de mettre sur pied un cadre de prototypage que nous appelons MEF (Malware Emulation Framework) qui sera utilisé pour créer des prototypes de logiciels malicieux. Ces prototypes serviront à explorer l'espace des caractéristiques des maliciels et seront comparés entre eux pour mieux comprendre les relations qui existent entre l'environnement, les caractéristiques et les indices de performances. Nous appelons "prototypes" les programmes générés par MEF parce que ce sont des agents qui émulent le comportement des maliciels mais qui ne peuvent être utilisés dans des vrais réseaux. Nous donnons plus de détails sur les mesures de sécurité utilisées pendant nos recherches à la fin de ce chapitre. L'évaluation de performance des maliciels à l'aide du cadre de prototypage permet

d'identifier les mécanismes de défense qui fonctionnent bien et de trouver les faiblesses des maliciels pour mieux défendre nos réseaux. De plus, la création de prototypes de maliciels dans un environnement contrôlé produit un milieu idéal pour évaluer la qualité des systèmes de détection d'intrusion présents sur le marché et d'observer leur capacité de détection de menaces inconnues. À l'aide de prototypes de maliciels, il sera aussi possible d'évaluer des systèmes de prévention d'intrusion et l'efficacité des dispositifs de murs pare-feu.

Le cadre de prototypage est un ensemble de classes et de fonctions qui sont utilisées pour créer rapidement des prototypes de maliciels. Le scientifique qui veut créer une expérience a simplement à créer une instance d'un maliciel et à spécifier ses caractéristiques et les paramètres associés. Plusieurs caractéristiques seront déjà programmées et les paramètres pertinents seront simplement spécifiés. Bien sûr, l'organisation des caractéristiques suit le modèle de la boucle OODA présenté au chapitre 3.

5.1.1 Architecture

Le cadre de prototypage est un ensemble de classes dans le langage de programmation Ruby [1]. Quand on veut créer une expérience avec un nouveau prototype de maliciel, on doit créer une classe qui hérite de `Malware`. Pour chacune des quatre phases de la boucle OODA, le chercheur doit spécifier quelles caractéristiques il donne au maliciel. Dans la phase d'initialisation, le chercheur spécifie les paramètres de chaque caractéristique utilisée. Le fonctionnement général du maliciel se trouve dans la classe `Malware` qui s'occupe de garder une trace d'exécution et de se déplacer dans la boucle *OODA*.

La figure 5.1 montre un exemple d'organisation des modules d'observation et des paramètres qui sont associés à chaque caractéristique. Dans cette figure, on voit que 3 fonctionnalités d'observation ont été implémentées : *TCPScan, UDPScan* et *Monitor*. Les détails de ces fonctionnalités sont expliquées à la section 5.1.2. Toutes les fonctions d'observation ont le même nom et retournent toutes une variable de type dictionnaire contenant les informations récoltées. Par exemple, un balayage pour identifier les ports TCP ouverts sur un réseau produira un dictionnaire contenant les adresses IP et les ports détectés comme étant ouverts. Chaque fonctionnalité d'observation nécessite

[1] http ://www.ruby-lang.org/

des paramètres différents. Ces paramètres sont spécifiés en argument à la fonction à l'aide d'un dictionnaire pour garder une uniformité entre les prototypes des fonctions. Les fonctionnalités d'orientation, de décision et d'action fonctionnent exactement de la même manière que celles d'observation. Les paramètres des fonctionnalités sont soit pré-programmés par le chercheur lors de l'initialisation, soit modifiés par le maliciel lors de sa phase de décision.

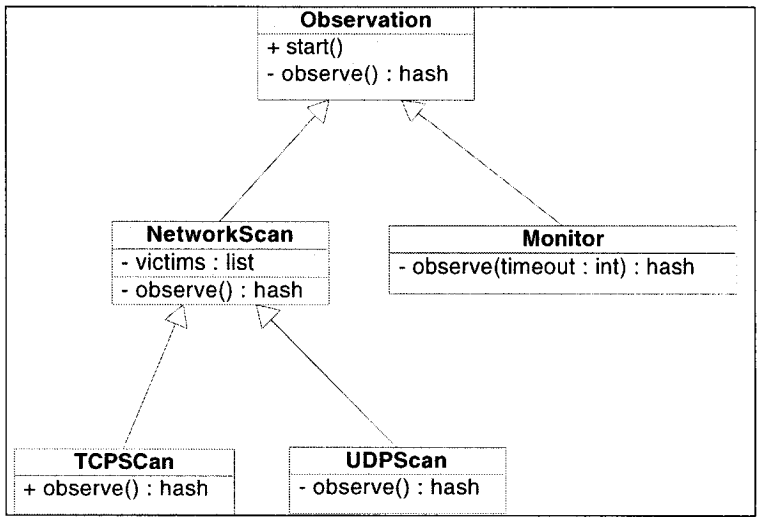

FIGURE 5.1 Création d'un module d'observation

Pour montrer le fonctionnement du cadre de prototypage, nous présentons quelques sections de code produites à l'aide du cadre de prototypage. Les lignes du listing 5.1 montrent la phase d'initialisation d'un prototype de maliciel. On commence par inclure les classes qui seront nécessaires lors de l'exécution. Par après, on implante le constructeur de la classe dans lequel on spécifie l'identifiant du maliciel. Cet identifiant sera utilisé pour garder la trace du maliciel lors de sa propagation. L'identifiant peut être un simple entier ou une chaîne de caractères, au choix du programmeur.

Pendant la phase d'initialisation, les valeurs par défaut des paramètres sont affectées. Elles sont toutes stockées dans un dictionnaire qui s'appelle `@memory`. Ce dictionnaire est un attribut de la classe `Malware` et est donc accessible à partir de toutes les méthodes des maliciels. Dans l'exemple, le programmeur détermine que la première observation qui serait faite par le maliciel sera un balayage aléatoire pour trouver les ports TCP ouverts. Les paramètres précisent que le balayage sera effectué sur le sous-réseau `'192.168.0.*'` sur les ports 21 (FTP), 22 (SSH) et 80 (HTTP). La

phase d'orientation sera une connexion au serveur IRC situé à l'adresse `192.168.1.1` et la communication se fera par le canal `#command_control`.

Listing 5.1 Phase d'initialisation

```
require 'Malware'
require 'observation/random_tcp_recon'
require 'orientation/IRC'

class IRCMalware < Malware
  def initialize(id = DEFAULT_ID)
    super

    @memory['observation_type'] = 'tcp_random_scan'
    @memory['tcp_random_scan_mask'] = '192.168.0.*'
    @memory['tcp_random_scan_ports'] = [21,22,80]

    @memory['orientation_type'] = 'connect_irc'
    @memory['irc_server'] = '192.168.1.1'
    @memory['irc_channel'] = '\#command_control'
  end
```

La partie de code du listing 5.2 montre l'utilisation des fonctionnalités de balayage TCP. L'utilisation de cette classe est très simple. On a simplement à appeler la méthode `start` avec l'attribut `@memory` qui contient les paramètres pertinents. Les résultats produits par cette fonctionnalité sont ajoutés à la variable `@memory`. On voit que la fonctionnalité d'observation sera seulement effectuée si le type d'observation contient la valeur `tcp_random_scan`. Dans les autres cas, aucune fonctionnalité d'observation n'est effectuée.

Listing 5.2 Phase d'observation

```
  def observe
        super

    if @memory['observation_type'] == 'tcp_random_scan' then
      RandomTCPRecon.start(@memory)
    end
  end
```

Pour la phase d'orientation, le prototype que nous décrivons se connecte à un serveur utilisant le protocole Internet Relay Chat (IRC). Vu que les paramètres ont été

affectés lors de l'initialisation, la seule chose qui est implémentée dans cette méthode est la création d'une instance d'un nouveau canal IRC et le démarrage de la communication. La fonctionnalité IRC permet d'échanger des informations entre les instances malicieuses et la personne qui les contrôle. Rien d'autre n'a à être implanté au niveau du maliciel puisque toutes les fonctionnalités de communication sont stockées dans la classe `IRC`. La section 5.1.2 donne plus d'informations sur les fonctionnalités implantées dans le cadre de prototypage.

Listing 5.3 Phase d'Orientation

```
1  def orient
2    super
3
4    if @memory['orientation_type'] == 'connect_irc' then
5      irc_channel = IRC.new
6      irc_channel.start(@memory)
7
8    end
9  end
```

Le maliciel cité en exemple n'effectue aucune décision de manière indépendante. Toutes les actions qu'il prend sont dictées par la voie de communications avec la personne qui le contrôle. C'est pourquoi les lignes du listing 5.4 montrent une méthode vide qui ne fait qu'appeler la méthode de la classe mère. Dans notre cas, c'est la méthode `decide` de la classe `Malware` qui est appelée.

Listing 5.4 Phase de décision

```
1  def decide
2    super
3  end
4  end
```

Aucune action n'a été implémentée dans ce maliciel de démonstration parce qu'il est utilisé pour étudier l'effet des communications entre maliciels. On voit à la section de code 5.5 que la seule action qui est posée est d'attendre 3 secondes à chaque fois qu'une itération de la boucle *OODA* est effectuée.

Listing 5.5 Phase d'action

```
1   def action
2     super
3     sleep 3
4   end
5 end
```

La dernière section qui doit être implémentée dans un prototype de maliciel est sa phase de lancement, c'est à dire le point d'entrée dans le code qui démarre l'exécution du maliciel. Les lignes 5.6 montrent que cette opération est simple. Le programmeur doit simplement créer une instance du prototype et la lancer à l'aide de la méthode `start`. La méthode `start` est montrée à la section de code 5.7. On voit qu'elle implémente la boucle *OODA* et garde en mémoire le nombre d'itérations effectuées.

Listing 5.6 Lancement du maliciel

```
1
2 ircmalware = IRCMalware.new
3 ircmalware.start
```

Listing 5.7 Méthode start

```
1   # This is the entry point of the program, start the while loop
2   def start
3
4     while telemetry(@memory, @cpt_iteration)
5       observe
6       orient
7       decide
8       action
9
10      @cpt_iteration = @cpt_iteration + 1
11    end
12
13    cleanup
14  end
```

Afin d'assurer une propagation rapide et représentative de celle d'un vrai maliciel, nous avons créer un utilitaire qui prend toutes les classes et modules nécessaires au bon fonctionnement d'un maliciel et les place dans un même fichier. Ce fichier est

ensuite facilement copié d'un système à l'autre sans besoin de garder une notion de l'emplacement du script dans le système de fichiers.

Plusieurs raisons nous ont poussés à utiliser le langage Ruby pour ce projet. Premièrement, le cadre de création de modules d'exploitation *metasploit* est aussi programmé en Ruby. Ce qui nous permet de rapidement utiliser ces modules d'exploitation et des les incorporer dans notre cadre. Le langage Ruby est un langage moderne et orienté objet. Son apprentissage est très rapide et sa syntaxe claire. De plus, c'est un langage interprété, ce qui fait que nos prototypes peuvent s'exécuter sur différentes systèmes d'exploitation sans efforts supplémentaires de développement.

5.1.2 Fonctionnalités déjà implémentées

Cette section donne un aperçu des fonctionnalités qui ont été implémentées dans le cadre de prototypage. Le tableau 5.1 montre un résumé des fonctionnalités et le reste de la section donne une description plus complète de chacune des fonctionnalités et des paramètres qui y sont associés.

TABLEAU 5.1 Fonctionnalités implémentées dans MEF

Observation	Orientation	Decision	Action
Balayage TCP	Communication	Simple	Infection (SSH)
Balayage UDP			Infection (msf)
Écoute passive			Propagation
Présence alliée			Auto-destruction

Balayage TCP. Parcour d'une liste d'adresses IP à la recherche de certains ports TCP ouverts. Cette technique de collecte d'information sur un réseau est la plus souvent utilisée par les pirates informatiques et les maliciels pour découvrir des services potentiellement vulnérables s'exécutant sur un ordinateur connecté au réseau. Cette fonctionnalité utilise un délai maximal de connexion d'une seconde, ce qui signifie que si un serveur ne répond pas après une seconde, il est considéré comme absent. Cette fonctionnalité prend en entrée une liste d'adresses IP à balayer ainsi qu'une liste de ports TCP à vérifier. La sortie est une liste de tuples IP et port qui sont ouverts.

Balayage UDP. Fonctionnalité très semblable au balayage TCP sauf qu'elle utilise le protocole UDP. Ce protocole est différent entre autres parce qu'il n'utilise pas de connexions complètes. Il est possible d'envoyer un paquet de données sur un port

UDP et que le serveur le traite sans jamais envoyer d'accusé réception. Le balayage se fait donc en utilisant les messages d'erreur ICMP qui sont renvoyés par un serveur lorsqu'un paquet est envoyé sur un port fermé. Ce type d'observation est moins fiable que le balayage TCP mais permet de découvrir des services qui n'utilisent pas le protocole TCP. Cette fonctionnalité utilise les mêmes paramètres que le balayage TCP.

Écoute passive. Configuration de l'interface réseau pour tomber en mode espion (*promiscuous* en anglais), permettant ainsi d'écouter les communications entre différents ordinateurs et de savoir quels ports sont utilisés. Cette fonctionnalité prend le nom de l'interface réseau sur lequel l'écoute sera effectuée en entrée et retourne une liste de tuples IP et port qui sont détectés comme ouverts. Un port est considéré ouvert quand on observe un paquet avec les drapeaux `SYN` et `ACK` activés, signifiant qu'une connexion TCP a été acceptée.

Présence d'une autre instance malicieuse. Quand le maliciel infecte un système, nous faisons en sorte qu'il copie le fichier contenant son programme dans le répertoire `/tmp`. La vérification de la présence d'une autre instance malicieuse est faite en utilisant le vecteur d'infection (décrit plus bas) et en vérifiant qu'aucun fichier malicieux n'est situé dans ce répertoire. Cette vérification est effectuée pour s'assurer que seulement une instance du maliciel peut s'exécuter sur un système pour ne pas saturer les ressources de la victime et fausser les résultats de l'expérimentation. Cette fonctionnalité ne prend pas de paramètre en entrée et retourne `vrai` si un fichier du maliciel se trouve déjà sur la victime potentielle.

Décisions simples et agressives. Le processus de décision qui est implanté dans le cadre de prototypage est très rudimentaire, à l'image des maliciels observés sur l'Internet d'aujourd'hui. Cette fonctionnalité prend en entrée un dictionnaire contenant toutes les informations récoltées lors de la phase d'observation et traitées lors de la phase d'orientation. À l'aide de ces informations, une décision est prise et retournée dans un dictionnaire (une version augmentée des informations reçues en entrée). Les décisions sont prises à l'aide de structure "si alors, sinon". Par exemple, si la phase d'observation nous indique un service vulnérable, alors on essaie de l'infecter. Si le maliciel a le droit d'exécution sur un système, alors il s'y copie et commence l'exécution d'une nouvelle instance.

Le processus de décision qui est implanté dans le cadre est très simple et pourrait être amélioré. Par exemple, il serait possible d'ajouter une processus de décision qui

utilise les réseaux de neurones pour choisir la meilleure action à prendre. Il serait aussi possible d'effectuer des décisions sur une liste d'action au lieu d'uniquement décider quelle sera la prochaine action comme c'est présentement le cas.

Infection par SSH. Fonctionnalité qui permet de gagner le droit d'exécution sur un système distant en connaissant la clé privée d'un utilisateur. Par cette fonctionnalité, le maliciel peut copier des fichiers sur sa victime et exécuter des commandes, principalement pour lancer l'exécution d'une nouvelle instance de maliciel. Cette fonctionnalité appelle directement le programme SSH qui doit être présent sur le système. Les paramètres en entrée sont l'adresse IP du système ainsi que le nom d'utilisateur à utiliser. La clé privée de l'utilisateur doit se trouver dans le répertoire `.ssh` pour être utilisée.

Infection à l'aide du cadre Metasploit. Afin d'ajouter une grande souplesse dans les vecteurs d'infection qui peuvent être utilisés par le cadre de prototypage, nous avons intégré la possibilité d'utiliser tous les modules d'infection présents dans la troisième version du logiciel libre Metasploit [2]. Metasploit est un cadre de développement de programmes d'exploitation de vulnérabilités logiciel qui contient des dizaines de programmes fonctionnels. Cette fonctionnalité prend un dictionnaire en entrée avec les informations nécessaires au fonctionnement du module d'exploitation. Les informations habituelles sont : le nom du module d'exploitation, l'adresse IP de la victime, la version du système d'exploitation attaqué ainsi que le nom de la charge active. Nous utilisons toujours la charge active "invite de commande" ("command shell" en anglais) pour simplement prendre le contrôle du système distant.

En plus d'utiliser les fonctionnalités de Metasploit, nous avons programmé un serveur vulnérable à une faille par débordement de tampon ainsi qu'un module Metasploit pour l'exploiter. Cette faille nous permettra d'émuler le comportement d'un maliciel qui exploite une faille inconnue dans un serveur.

Propagation. C'est au cours de cette action que le maliciel crée une copie de son code et lance son exécution sur la machine distante. Avant d'initier la propagation, le maliciel donne un nouvel identifiant unique au code qu'il copie. Une fois la copie locale créée, elle est envoyée par le biais du réseau vers la victime avant d'être exécutée à l'aide du canal de commande créé lors de la phase d'infection (ce canal de commande est souvent un identifiant de "socket" réseaux). Il est donc nécessaire que la propagation se fasse après la phase d'infection. Cette fonctionnalité prend en entrée

[2]http ://www.metasploit.org/framework

le canal de communication créé lors de l'infection et retourne l'identifiant du nouveau maliciel qui a commencé son exécution sur la victime. La phase de propagation est délicate parce qu'il faut s'assurer que l'exécution est bien débutée sur la victime sans bloquer l'exécution du maliciel père. Nous avons décidé d'implémenter cette phase dans un "thread" différent pour s'assurer du parallélisme de ces opérations.

Autodestruction. Cette fonctionnalité indique au maliciel d'effacer les fichiers qu'il avait écris sur le disque et de stopper son exécution. Cette fonction est appelée sur ordre de la personne qui conduit l'expérience par le biais du canal de communication.

5.1.3 Mesures de sécurité

Toute expérimentation avec des logiciels malicieux doit être faite avec les plus grandes précautions pour s'assurer qu'aucune fuite n'est possible vers l'Internet et que l'expérience reste sous le contrôle du chercheur pendant toute son exécution. Pour s'assurer de la sécurité des expériences, nous avons inclus plusieurs dispositifs de sécurité dans les prototypes étudiés.

La première mesure de sécurité implantée dans les prototypes est un mécanisme de télémétrie. Ce mécanisme établit une connexion vers un serveur de télémétrie à chaque itération de la boucle OODA. Si le serveur ne répond pas à la requête par la chaîne de caractère "OK", le maliciel arrête son exécution. Pour terminer rapidement une expérience, le chercheur peut donc simplement arrêter le serveur de télémétrie et toutes les instances malicieuses arrêteront de s'exécuter et de se propager. En plus d'amener une sécurité lors des expériences, la télémétrie est utilisée pour accumuler des statistiques sur le nombre de systèmes infectés dans le réseau et la vitesse de propagation. Dans le cas d'expériences dans un environnement moins sécurisé qu'un réseau dédié, il serait possible d'augmenter la robustesse de la télémétrie avec de la cryptographie. Par exemple, les agents malicieux pourraient s'authentifier auprès du serveur à l'aide d'une procédure "challenge - response" à clé privée, assurant ainsi l'authenticité mutuelle des agents et du serveur.

Même si la propagation des maliciels est contrôlée à l'aide de la télémétrie, on doit s'assurer qu'aucun service légitime ne sera perturbé par le trafic généré par les instances malicieuses. Pour ce faire, nous utilisons un environnement complètement contrôlé à l'aide d'un serveur VMWare. Le réseau est constitué de machines virtuelles qui peuvent être rapidement éteintes à l'aide de la console VMWare. Cette confi-

guration nous permet aussi d'utiliser des commutateurs logiciels garantissant ainsi qu'aucun trafic malicieux ne quitte le réseau virtuel.

Toutes les failles de sécurité qui sont exploitées par les agents générés à l'aide de MEF sont toujours créées par les expérimentateurs pour s'assurer qu'aucune fuite vers un système légitime ne soit possible. C'est le cas, entre autres, du faux serveur TCP que nous avons programmé accompagné d'un module d'exploitation pour le cadre Metasploit.

Le mécanisme d'infection que nous jugeons le plus sécuritaire est celui qui utilise les clés publiques et privées du protocole SSH. Cette technique pour gagner le contrôle d'une victime est sécuritaire parce qu'il garantit que l'agent malicieux pourra uniquement infecter les systèmes qui ont la clé publique correspondant à sa clé privée. Vu que le protocole SSH permet d'exécuter des commandes et de transférer des fichiers dans le même flot de données, ce choix amène aussi plus de stabilité au système cible.

Finalement, le fait d'utiliser le langage Ruby peut être vu comme une mesure de sécurité. Étant donné que les maliciels créés seront des programmes interprétés, seules les stations ayant un interpréteur Ruby pourront être infectées ; cet interpreteur n'est pas très répandu ni déployé par défaut sur la plupart des systèmes qu'on retrouve sur l'Internet.

À ce point, il est légitime de se demander quels seraient les efforts nécessaires à un programmeur malicieux pour récupérer le cadre MEF et l'utiliser pour créer des maliciels fonctionnels. Nous pensons que notre cadre est très utile à des fins de recherche mais qu'aucune des fonctionnalités présentement implantées n'est nouvelle et ne serait attrayante pour un utilisateur malveillant. De plus, des modifications majeures devraient être effectuées au code pour désactiver les mécanismes de sécurité comme la télémétrie. Nous pensons qu'un utilisateur malicieux choisirait sans hésitation d'utiliser d'autres programmes plus complets, programmés dans un langage plus portable et disponibles sur l'Internet[3] avant de s'intéresser à notre création.

[3]http ://www.rootkit.com/

5.2 Expériences de propagation par rapport au filtrage

Cette section décrit les expériences pratiques que nous avons exécutées au sein du laboratoire de sécurité des réseaux. La section 5.2.1 décrit le contexte technique de l'expérience et la section 5.2.2 résume les résultats obtenus.

Cette section utilise la même nomenclature que la section 4.2.3, c'est à dire que nous considérons un réseau séparé en deux parties. Ces parties sont reliées par des systèmes qui sont connectés aux deux parties simultannément. On comprend que pour qu'un maliciel puisse infecter un système dans la deuxième partie du réseau, il doit premièrement infecter la passerelle qui assure la connexion. Nous avons effectué nos expériences avec 30 systèmes. Nos expériences ont été faites avec trois scénarios de configuration différents :

- Scénario 1 : aucun filtrage.
- Scénario 2 : filtrage bas (4 systèmes connectés aux deux sous-réseaux).
- Scénario 3 : filtrage élevé (1 système connecté aux deux sous-réseaux).

La figure 5.2 montre la configuration du réseau pour le scénario 3. On voit que le premier sous-réseau est constitué de 14 systèmes, qu'un système est connecté aux deux réseaux et que le deuxième sous-réseau compte 15 systèmes.

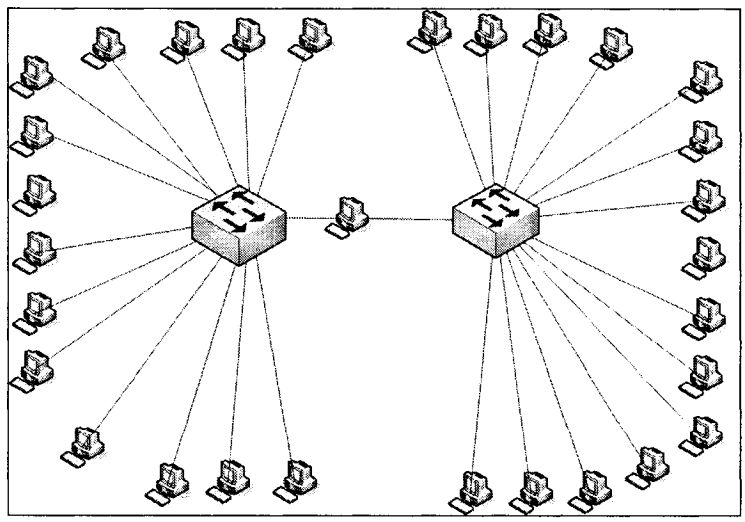

FIGURE 5.2 Topologie du réseau avec une interconnexion

5.2.1 Contexte technique

Toutes nos expériences ont été effectuées dans un réseau virtuel peuplé de machines virtuelles s'exécutant sur un système VMWare ESX 3 appartenant au Laboratoire de Sécurité des Réseaux Informatiques (LSRI) de l'École Polytechnique de Montréal. Pour contrôler les machines virtuelles, nous les avons configurées pour qu'elles utilisent trois interfaces réseaux. Deux de ces interfaces sont utilisées pour les réseaux de tests et la troisième est utilisée comme réseau de contrôle. Le réseau de contrôle permet de garder une connexion avec tous les systèmes pour les configurer et récolter des informations de télémétrie malgré les mécanismes de filtrage présents entre les autres réseaux.

TABLEAU 5.2 Caractéristiques des machines virtuelles

Système d'opération	Debian Linux
Mémoire vive	32 méga octets
Capacité de disque	750 méga octets
Interpreteur Ruby	Version 1.8.2
Serveur SSH	OpenSSH 3.8.1

Les machines virtuelles utilisées ont les caractéristiques décrites dans la table 5.2. Pour implémenter le filtrage d'une façon simple et efficace, nous avons fait en sorte qu'aucun système ne fasse suivre des paquets qui ne lui sont pas destinés. Les seuls systèmes qui peuvent établir des connexions vers les deux sous-réseaux sont ceux qui ont une interface sur chacun de ces réseaux.

Pour changer les configurations réseau des systèmes sans avoir à utiliser la console VMWare, nous avons utilisé les serveurs SSH installés sur les machines virtuelles avec des scripts Ruby. Ces scripts génèrent automatiquement les fichiers de configuration réseau pour les machines virtuelles en fonction de leur identifiant (chiffre entre 1 et 30) et copient les fichiers de configuration par SSH. Quand les configurations sont en place, le script redémarre la machine virtuelle pour s'assurer que toutes les configurations prennent automatiquement effet.

Lors d'une expérimentation, la personne qui dirige l'exécution doit lancer un serveur de télémétrie pour contrôler l'expérience et récolter les informations sur la propagation. Elle doit aussi s'assurer que la configuration du réseau est bonne. Quand tout est prêt, le chercheur lance l'exécution du maliciel sur une station dans le premier sous-réseau. Quand le chercheur constate que tous les systèmes observés sont infectés,

il peut mettre fin à l'expérience en terminant l'exécution du serveur de télémétrie. Un script de nettoyage des stations doit ensuite être lancé pour effacer toutes les traces laissées par les maliciels avant de pouvoir conduire une nouvelle expérience.

La section 5.2.2 résume les résultats que nous avons obtenus lors des phases d'expérimentation pratiques.

5.2.2 Résultats

Cette section résume les résultats expérimentaux récoltés dans le cadre de nos recherches. Chaque expérience a été exécutée cinq fois pour s'assurer de la précision des données. Le nombre de système infectés a été calculé à partir des journaux générés par le serveur de télémétrie décrit au chapitre précédent. Nous avons produit une courbe du nombre de systèmes infectés avec une granularité de trois secondes, c'est à dire que nous évaluons le nombre de systèmes infectés dans le réseau à chaque trois seconde à partir des journaux de télémétrie.

Les courbes de propagation de la première expérience sont présentées à la figure 5.3. Les cinq lignes représentent le nombre de systèmes infectés en fonction du temps (en secondes) pour chacune des cinq expériences exécutées. Les points bruns représentent la moyenne des cinq expériences.

La figure 5.4 montre les résultats et la moyenne des cinq expériences du scénario 2. Dans ce deuxième scénario, on voit que le nombre de systèmes infectés en fonction du temps varie plus d'un essai à l'autre. Nous pensons que ces différences sont dues au filtrage qui fait en sorte qu'il est plus difficile pour les maliciels de gagner accès à la deuxième portion du réseau. Dans le cas de la troisième expérience, on voit que la deuxième partie du réseau a été rapidement infectée. D'un autre côté, l'essai 2 montre que quelques minutes peuvent s'écouler avant que la deuxième partie du réseau soit attaquée par les maliciels.

La figure 5.5 montre les résultats des 5 phases de tests ainsi que la moyenne des résultats observés pour le dernier scénario impliquant seulement un système interconnectant les deux sous-réseaux. On voit que les vitesses de propagation varient beaucoup entre elles. Cette variation est principalement notable quand les 15 premiers systèmes sont infectés. Les plateaux observés avant l'infection du seizième système pour les essais 3 et 4 sont directement liés au filtrage mis en place entre les deux sous-réseaux de l'environnement.

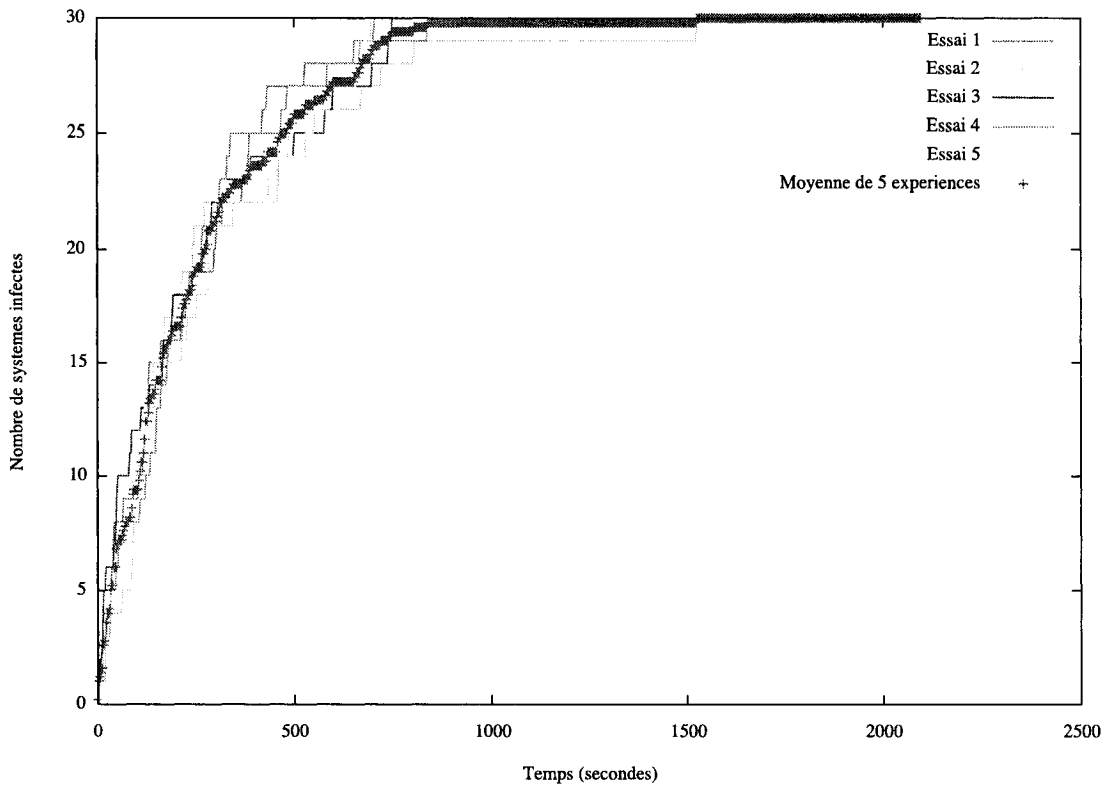

FIGURE 5.3 Résultats scénario 1

5.2.3 Analyse et interprétation des résultats

Le tableau 5.3 montre certains résultats numériques tirés des trois scénarios de configuration de réseau. Comme on a pu le voir à la figure 5.3, la variance entre les expériences est relativement faible (0.5687). L'écart entre le temps le plus court pour infecter tous les systèmes et le plus long est dû au fait qu'il est parfois difficile de trouver les derniers systèmes à infecter parce qu'à chaque essai d'infection, les maliciels ont seulement une chance sur 256 (nombre d'adresses disponibles) de générer aléatoirement la bonne adresse (celle du seul système qui n'est pas encore infecté).

Les résultats numériques du deuxième scénario montrent que la variance est beaucoup plus grande dans cette expérience que dans la précédente. On voit aussi que le temps moyen pour infecter les 30 systèmes vulnérables a augmenté de plus de huit minutes, démontrant que le filtrage a comme effet de réduire la vitesse de propagation de certains maliciels. Les résultats numériques nous montrent aussi que le temps moyen pour infecter dix systèmes est un peu plus bas que celui observé pour le pre-

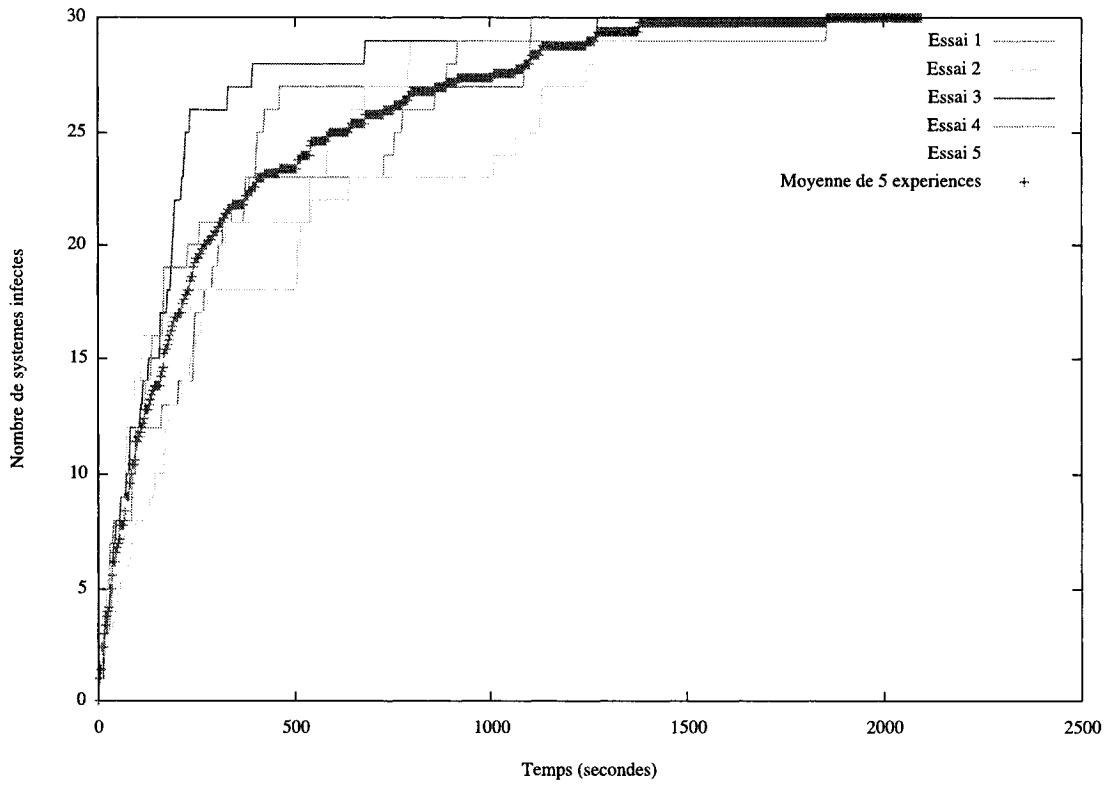

FIGURE 5.4 Résultats scénario 2

mier scénario. Nous pensons que la différence n'est pas significative et qu'il est normal que les temps d'infection des premiers systèmes soient similaires puisque le filtrage entre seulement en jeu lorsque les maliciels attaquent la deuxième partie du réseau.

Dans les résultats du troisième scénario, on constate que le temps moyen pour infecter tous les systèmes est de presque 23 minutes, comparativement à 15 minutes dans une situation où aucun filtrage n'est présent. D'un autre côté, on note aussi que le temps moyen d'infection de tous les noeuds du réseau est légèrement plus court que lors du scénario précédent. La différence est d'à peine trente secondes pour vingt-trois minutes. Nous pensons que cette différence est minime mais elle montre tout de même que le nombre de systèmes interconnectés entre deux réseaux n'a pas autant d'effet qu'on aurait pu le penser. Il faut par contre noter que presque tout au long de l'expérience, le nombre de systèmes infectés est resté inférieur à celui du scénario 2. Le temps moyen pour infecter 20 systèmes est presque deux fois plus long qu'à l'expérience 2 tandis que le temps pour infecter 25 systèmes est de 969 contre 639

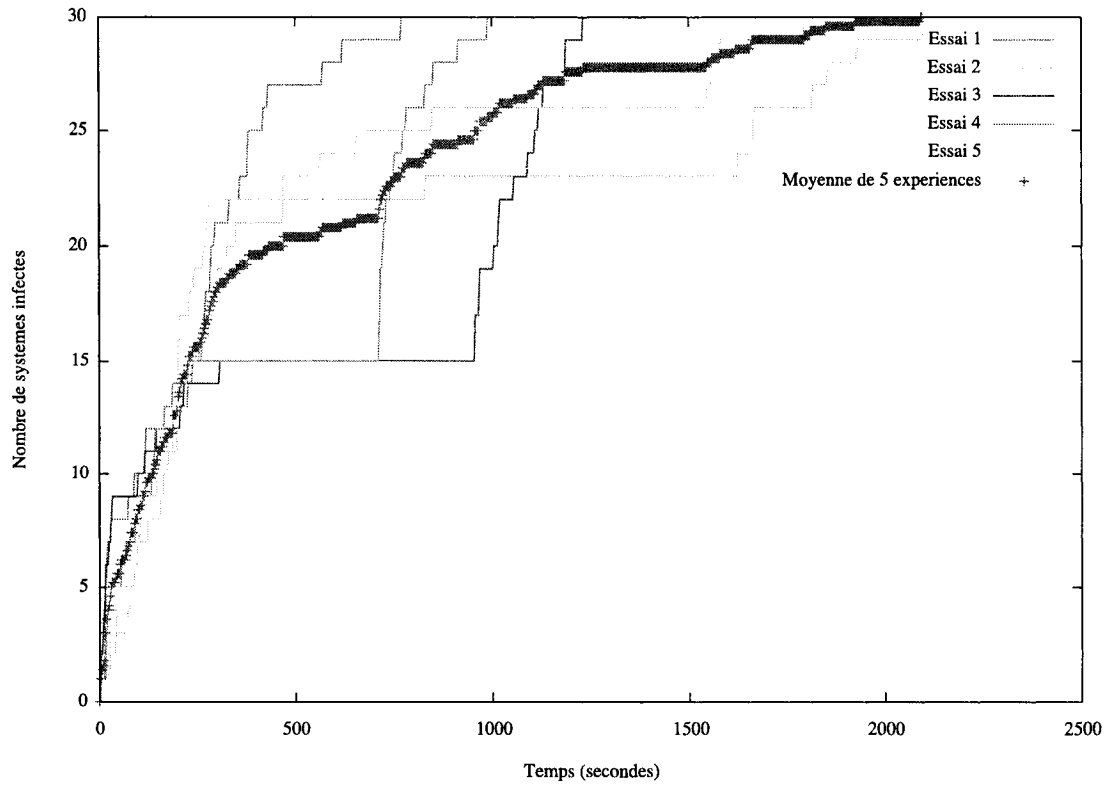

FIGURE 5.5 Résultats expérience 3

secondes dans le cas précédent.

5.3 Comparaison des résultats

La figure 5.6 montre la courbe de la moyenne des résultats du premier scénario et la courbe obtenue avec le modèle théorique. Les deux autres lignes représentent l'intervalle de confiance à 95% pour la moyenne des système infectés pendant l'expérience pratique.

Cette figure montrent que les deux courbes ont les mêmes tendances mais pas la même pente. Le modèle théorique a un départ plus lent de propagation mais augmente plus rapidement que nos observations pour les 100 secondes qui suivent. Malheureusement, la courbe théorique tombe rarement à l'intérieur des intervalles de confiance calculés autour de la moyenne pratique. Vu que notre modèle vise à représenter la relation entre le filtrage et la vitesse de propagation, nous ne pensons pas qu'il soit

TABLEAU 5.3 Résultats numériques des expériences pratiques

Temps en (s)	Scénario 1	Scénario 2	Scénario 3
Variance moyenne (sur 5 expériences)	0.5687	3.87	10.82
Plus court pour infecter 30 systèmes	696	1107	771
Plus long pour infecter 30 systèmes	1527	1857	2094
Moyenne pour infecter 30 systèmes	903.6	1400	1377
Moyenne pour infecter 25 systèmes	483	639	969
Moyenne pour infecter 20 systèmes	279	279	471
Moyenne pour infecter 10 systèmes	108	87	141

nécessaire de changer notre modèle pour qu'il représente mieux une situation de réseau sans filtrage. Les modèles épidémiologiques présentés au chapitre 2 sont mieux adaptés à ce type de situation.

À la figure 5.7, on voit la courbe moyenne des cinq expériences effectuées pour le deuxième scénario avec ses intervalles de confiance à 95% et la courbe générée par le modèle théorique. On voit que le modèle mathématique diffère de façon importante de la réalité pour modéliser la vitesse d'infection des 15 premiers systèmes. Ce stade passé, la courbe théorique est à l'intérieur de l'intervalle de confiance à 95%. On constate que dans ce cas, la variance est beaucoup plus importante, ce qui fait en sorte que les intervalles de confiance sont plus larges.

Nous pensons que ces résultats sont satisfaisants puisqu'ils sont relativement précis pour prédire la vitesse de propagation au pire moment de l'infection, c'est à dire là où les prédictions sont les plus importantes. Par contre, les courbes n'ont pas vraiment la même trajectoire, ce qui nous laisse penser que notre modèle est probablement mieux adapté pour prédire le comportement d'une épidémie ayant lieu dans un gros réseau (plusieurs milliers de systèmes).

On peut voir les courbes de la moyenne, des intervalles de confiance à 95% et du modèle théorique pour le troisième scénario à la figure 5.8.

Comme pour le scénario précédent, la pente initiale est plus faible pour le modèle théorique que le comportement observé. La croissance du nombre de systèmes infectés dans le modèle théorique est aussi plus rapide que celle de nos résultats pratiques. Les tendances des deux courbes sont semblables mais elles n'ont pas la même pente, ce qui nous laisse croire que notre modèle est adéquat pour décrire le comportement d'une infection de maliciels mais mieux adapté à des échantillons de plus grande taille.

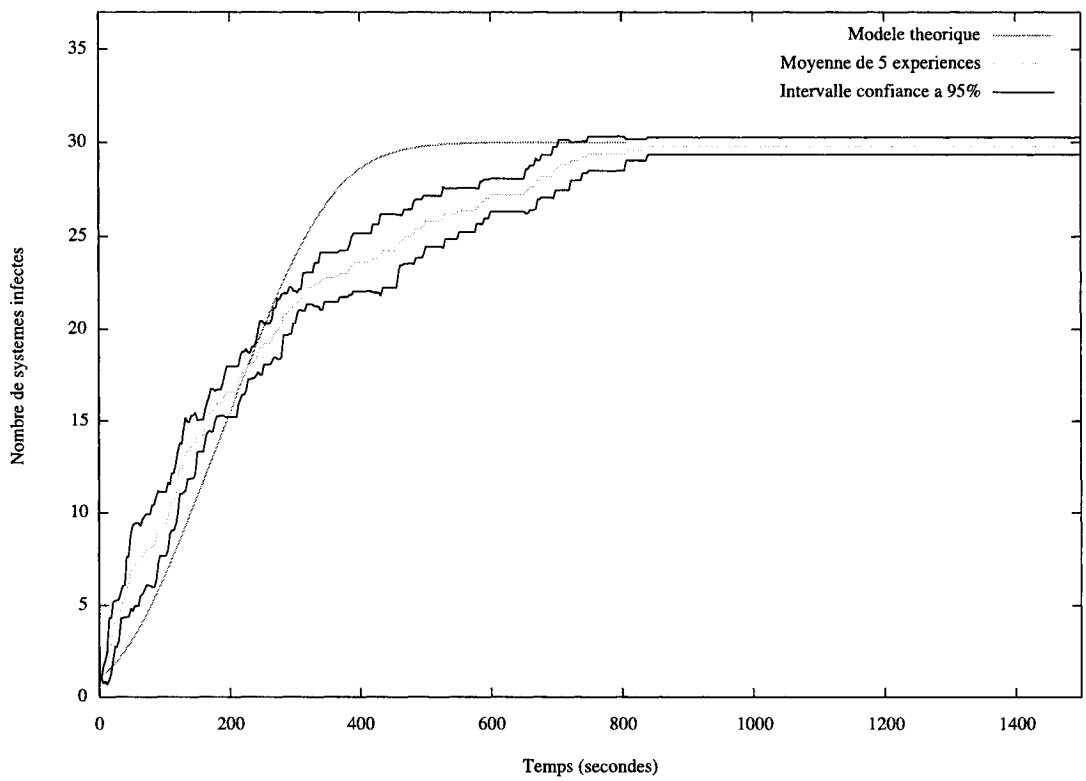

FIGURE 5.6 Comparaison des résultats du scénario 1

Pour bien visualiser l'effet relatif du filtrage sur les performance du maliciel, nous avons généré un graphe (présenté à la figure 5.9) qui montre les résultats pratiques recueillis lors de nos trois expériences. Cette figure nous permet de constater que le filtrage du trafic entre les parties d'un réseau ralentit effectivement la vitesse d'infection de certains maliciels.

Les comparaisons entre les résultats théoriques et pratiques indiquent que le modèle que nous avons développé donne une bonne représentation de la réalité, malgré le fait que la vitesse prévue est un peu plus lente pour les premiers systèmes a être infectés. Cette bonne représentation est en partie due au délai d'une seconde nécessaire à la propagation et au délai d'une seconde qui arrive quand un système ne répond pas à une tentative de connexion SSH. Dans le cas où un maliciel se propage par un service qui a des délais plus longs, on devrait considérer des temps différents pour une connexion manquée et pour une infection réussie. Le fait que notre fichier prend une seconde à être transféré et exécuté nous permet de faire abstraction de cette subtilité.

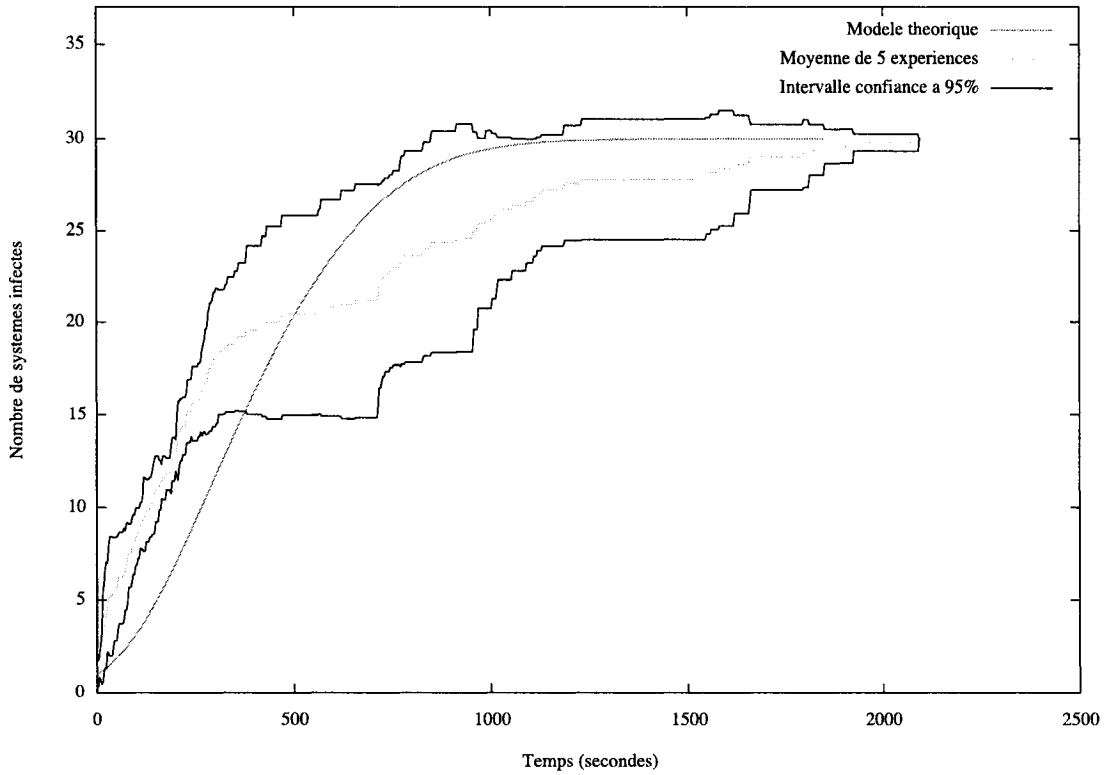

FIGURE 5.7 Comparaison des résultats du scénario 2

Parmi les trois expériences qui ont été comparées, le modèle s'applique moins bien au cas où aucun filtrage n'est effectué sur le réseau (scénario 1). Nous pensons que cette différence n'est pas très importante puisque le but premier de notre modèle mathématique est de comprendre la relation entre le filtrage entre des réseaux et la vitesse de propagation. Le scénario 1 est utile pour faire des comparaisons et montrer que le filtrage est efficace mais n'est pas représentatif des situations que nous voulons modéliser. Dans les trois scénarios d'expérience que nous avons étudié, la vitesse de propagation est plus lente au début de l'infection. Nous pensons que cette lenteur est due au manque de doubles transitions dans notre modèle. Ces doubles transitions permettraient de modéliser la propagation de plusieurs instances de maliciel dans un interval de temps. Nous avons décidé de ne pas considérer les doubles et triples transitions dans notre modèle théorique parce qu'ils rendaient le modèle plus complexe et plus difficile à résoudre.

Une autre faiblesse de notre comparaison est le petit nombre d'essais que nous

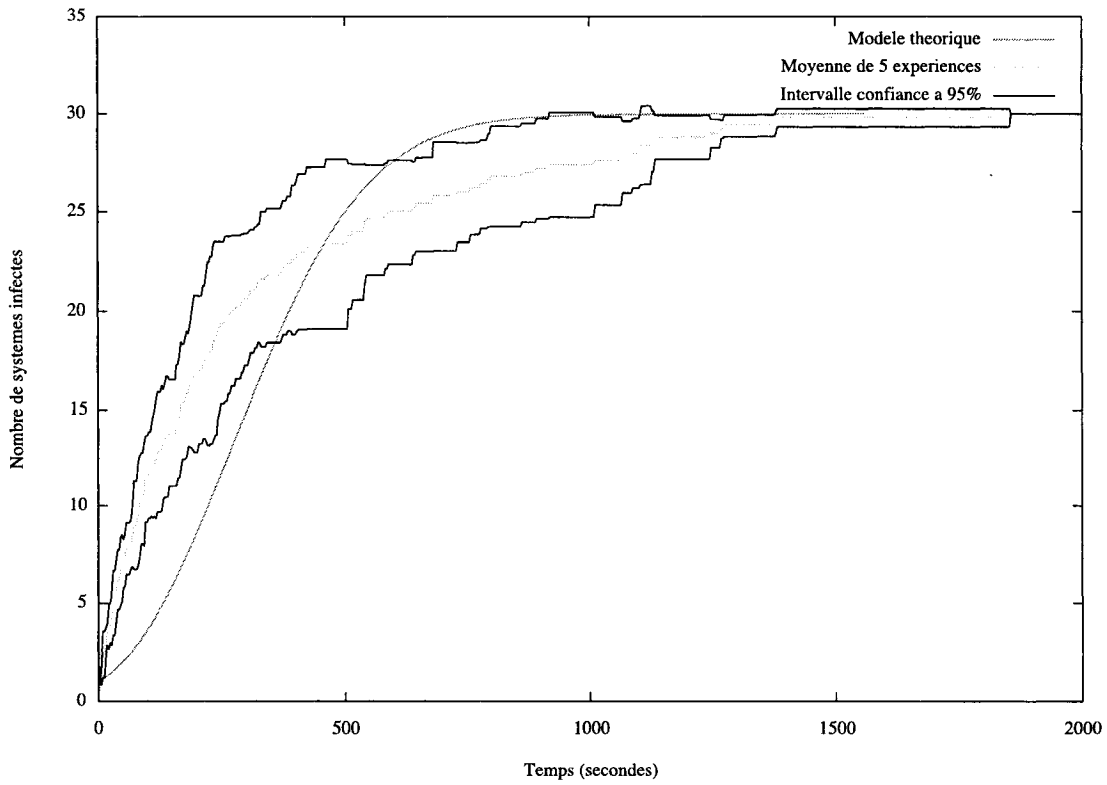

FIGURE 5.8 Comparaison des résultats du scénario 3

avons effectués pour chaque scénario. Des contraintes de temps et de ressources matérielles ont fait en sorte que seulement cinq essais ont pu être effectués par expérience. Dans des conditions idéales, il faudrait effectuer trente essais pour que nos données aient une valeur statistique significative ; plus concrètement, ceci nous aurait probablement permis de réduire la taille des intervalles de confiance. Pour être en mesure d'effectuer un aussi grand nombre d'essais, il faudrait automatiser complètement le processus d'expérimentation qui est présentement seulement automatisé en partie. L'arrêt de l'expérience et le nettoyage des systèmes infectés dans l'environnement de tests sont encore faits manuellement (en lançant les scripts et commandes pertinents au bon moment et dans la bonne séquence).

L'utilisation du logiciel de virtualisation VMWare ESX pour émuler le comportement de plusieurs systèmes sur un réseau peut être vue comme une limitation à notre travail. En effet, les ressources du système hôte ne sont pas nécessairement allouées de façon équitable entre les machines virtuelles. Ce partage des ressources peut donc

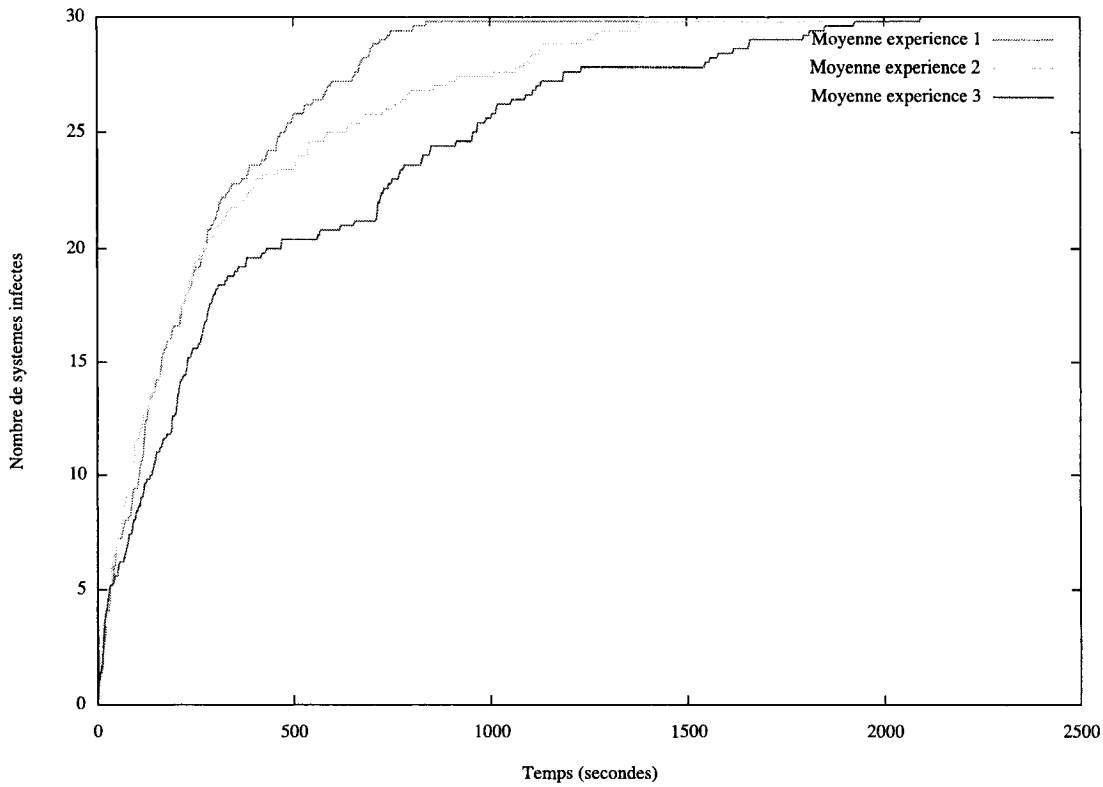

FIGURE 5.9 Comparaison des résultats des trois scénarios pratiques

avoir un impact important sur la vitesse de propagation que nous tentons d'observer. Nous pensons que la virtualisation n'a pas eu d'effet négatif sur les résultats de nos expériences. Les ressources du serveur VMWare n'ont pas été utilisées à plus de 40% des processeurs et 60% de la mémoire vive. Les machines virtuelles n'ont donc pas été mises en attente de ressources supplémentaires pendant nos expériences. Cela nous laisse croire que des résultats très similaires seraient observés dans un réseau n'utilisant pas la virtualisation.

Le modèle mathématique que nous avons développé et vérifié à l'aide de ces expériences est un outil qui nous permet d'effectuer des recommandations sur la configuration du filtrage réseau malgré qu'il semble mieux adapté à l'étude de gros réseaux. Même si le filtrage est une contre-mesure de sécurité connue et acceptée, notre modèle permet maintenant de quantifier ces mesures en calculant, par exemple, l'avantage d'exclure une machine des interconnexions sur la vitesse de propagation d'un ver informatique. Ceci est très important car il est connu que l'exclusion de

connexion d'un système aura potentiellement un impact en terme de performance et de fiabilité du réseau.

CHAPITRE 6

Conclusions

Dans ce mémoire, nous avons d'abord revu les différents efforts de caractérisation et d'évaluation de performances des logiciels malicieux. Nous avons revu les principaux courants de pensée dans la lutte à ces logiciels nuisibles. Cette étude de la littérature a permis de mettre en lumière la faiblesse des solutions de détection présentement utilisées. Nous croyons que cette faiblesse est due au fait que nous ne savons pas exactement quelle menace nous aurons à combattre dans le futur, ce qui justifie nos efforts pour mieux comprendre les maliciels et les facteurs qui influencent leurs performances.

Nous avons présenté deux modèles utiles pour l'étude des maliciels. Le premier est un modèle à trois facteurs qui permet d'exprimer la relation entre les indices de performances des maliciels, leur environnement et leurs caractéristiques. Le deuxième modèle est la boucle OODA, introduite par le colonel Boyd pour décrire le comportement des unités militaires. La boucle OODA s'applique bien aux maliciels et nous aide à décrire leurs caractéristiques. En plus des deux modèles, nous avons présenté une liste d'indices de performance correspondants à des objectifs pour la création et l'utilisation de logiciels malicieux.

Dans le quatrième chapitre, nous avons exposé l'étude de la relation entre un objectif de performance concret, la vitesse de propagation, et une caractéristique concrète de l'environnement, le filtrage réseau. Cette étude est une application de nos modèles théoriques que nous avons jugé pertinente puisqu'elle est utile pour sécuriser un réseau, nous aurions pu choisir d'autres indices de performance, d'autres caractéristiques de l'environnement ou même étudier une différente relation. Ce chapitre présente un modèle mathématique basé sur les processus de Markov pour exprimer la vitesse de propagation d'un maliciel dans un réseau filtré.

Le dernier chapitre présente une description technique de toutes les procédures effectuées pour observer les performances de maliciels dans un environnement contrôlé. Pour opérer nos expériences, nous avons bâti un cadre de prototypage que nous avons nommé MEF (*Malware Emulation Framework*) visant à créer des agents qui émulent

le comportement de maliciels. Nous avons ensuite créé un réseau de machines virtuelles où faire évoluer ces agents. Nous avons conduit trois types d'expériences (avec cinq exécutions chacune) pour étudier la vitesse de propagation d'un maliciel sous trois scénarios différents de configuration de réseau : un réseau sans filtrage, deux sous réseaux avec quatre système interconnectés et deux sous réseaux avec seulement un ordinateur assurant l'inter-connexion. C'est dans ce dernier chapitre que nous comparons les résultats théoriques et pratiques obtenus. Nous pensons que les résultats sont concluants et que les résultats théoriques sont relativement proches de la réalité observée.

Nos recherches portent sur un sujet épineux. La plupart des concepts exposés peuvent être utilisés autant par les intervenants qui veulent sécuriser leurs infrastructures que par les attaquants qui veulent apprendre à mieux pénétrer leurs réseaux cible. Nos recherches ont été effectuées après plusieurs réflexions et discussions éthique. Nous avons consulté des compagnies qui fournissent des logiciels d'anti-virus pour connaître toutes les techniques de sécurité que nous devions implanter et les limites que nous ne devons pas franchir. Ce sont ces discussions éthiques qui nous ont poussées à seulement utiliser des scripts avec des mécanismes de télémétrie pour effectuer nos expériences. Nous nous assurrions ainsi que les spécimens créés pour nos expériences ne posent aucun danger pour un ordinateur normal. Les techniques développées ici ne sont donc pas facilement utilisables par des programmeurs malveillants.

Les recherches présentées dans ce mémoire ont permis d'apporter les contributions suivantes au domaine de la sécurité informatique :

1. Modèle à trois dimensions pour décrire les performances des maliciels
2. Utilisation du modèle de la boucle OODA pour décrire les maliciels
3. Identification de faiblesses dans la conception des maliciels actuels et prévision de certaines évolutions, principalement dans les facultés d'orientation et de décision.
4. Liste des indices de performance des maliciels en fonction de leurs objectifs
5. Modèle mathématiques basé sur les chaînes de Markov pour décrire la propagation d'un maliciel dans un réseau filtré.
6. Cadre de prototypage pour faciliter l'étude de performance des maliciels
7. Méthodologie d'expérimentation pour effectuer des tests d'infection de maliciels dans un environnement contrôlé.

Les quatre premières contributions faites dans le cadre de ce mémoire ont été publiées au printemps 2006 dans une conférence sur les maliciels et l'organisation en essaim dans un article intitulé "Optimising Malware" (Bureau et Fernandez, 2006). Dans cet article, nous avons étudié les motivations derrière la création de logiciels malicieux et exposé les techniques d'optimisation qui pourraient être utilisées par les créateurs de maliciels pour augmenter les performances de leurs outils.

Le fait de prouver que le filtrage a pour effet de ralentir la propagation des logiciels malicieux n'est pas en soi une nouvelle contribution ; ceci était déjà connu du moins de façon intuitive par les experts dans le domaine. Par contre, ce qui est très remarquable et unique dans ce domaine, c'est que nous soyons arrivé à modéliser et vérifier expérimentalement cette intuition tout en lui donnant une expression **quantitative** concrète. De plus, toutes les démarches qui ont été effectuées pour élaborer et opérer un environnement contrôlé d'observation d'infection de maliciels seront certainement très utiles pour la communauté de chercheurs en sécurité informatique. La méthodologie de tests accompagnée du cadre de prototypage MEF décrit au chapitre 5 de ce mémoire sont des outils qui permettront aux chercheurs d'étudier plus facilement et efficacement les effets de certaines caractéristiques des maliciels et de l'environnement sur leurs indices de performance.

Finalement, la création d'un modèle théorique pour décrire l'effet du filtrage réseau sur la vitesse de propagation des maliciels est une addition aux techniques de modélisations déjà développées et permet de formuler des recommandations éclairées lors de l'élaboration des mécanismes de défense réseau et du développement des plans de récupération face à un désastre.

Nos recherches ont mené à la conception d'outils pour optimiser une caractéristique de l'environnement qui est le filtrage réseau. Grâce à ces outils, nous pouvons optimiser les systèmes de défense pour mieux répondre aux menaces posées par les maliciels en ralentissant le plus possible la vitesse de leur propagation, tout en gardant la performance et la fiabilité du réseau à des niveaux acceptables.

Nos recherches, quoi que fructueuses, n'ont fait qu'effleurer l'étude des logiciels malicieux. Ce domaine de recherche change très rapidement pour s'adapter aux nouvelles menaces et nous sommes convaincus que beaucoup peut être fait dans cette direction.

Nos travaux sur la modélisation théorique de l'effet du filtrage sur la propagation de maliciels pourraient être améliorés en tenant compte des topologies de réseau

qui ont plus que deux sous réseaux. Pour ce faire, il faudrait changer le système d'identification des états et augmenter la complexité des calculs nécessaires à l'obtention de résultats. Cette amélioration permettrait d'élaborer des recommandations plus générales et applicables à un plus grand nombre de topologies. Notre modèle théorique se base sur les modèles de Markov et sur le fait que les événements d'arrivée suivent une distribution de Poisson. Malgré le fait que les résultats pratiques montrent que notre modèle s'approche de la réalité, il serait intéressant d'explorer d'autres lois d'arrivée pour s'assurer que notre choix est judicieux. Finalement, le modèle pourrait aussi être amélioré en y incorporant les probabilités de multiples transitions dans un même intervalle de temps pour le rendre plus représentatif des comportements observés en laboratoire et sur l'Internet.

Les expériences que nous avons effectuées permettent de valider notre modèle théorique mais seraient plus intéressantes si un plus grand nombre d'exécutions avait été effectué. Il serait valable d'automatiser complètement le processus d'expérimentation pour que les exécutions puissent être effectuées sans que la présence d'un expérimentateur soit requise. En plus d'effectuer un plus grand nombre de tests par expérience, nous pensons qu'il serait intéressant d'effectuer des expériences avec des réseaux beaucoup plus volumineux que celui que nous avons utilisé. Un réseau de quelques milliers de noeuds permettrait de mieux observer les tendances parce que les expériences seraient plus longues et les résultats plus précis.

Le cadre de prototypage que nous avons construit et les méthodologies d'expérimentation que nous avons mises de l'avant ouvrent la porte à une très grande variété d'expériences qui n'ont pas encore été tentées. Il serait intéressant de créer une série d'expériences générales impliquant la propagation de maliciels dans un réseau de milliers de machines virtuelles. Ce réseau serait équipé de systèmes de défense commerciaux comme des systèmes de détection et de prévention d'intrusion. La batterie d'expériences pourrait être utilisée pour tester les performances des équipements défensifs et émettre des recommandations sur leur configuration et même leurs défauts possibles. En plus d'émettre des recommandations, les expériences permettraient d'aider les entreprises à faire un choix judicieux lors de l'acquisition de systèmes de défense en sachant exactement quelles sont les performances des systèmes dont ils font l'acquisition. Les nouvelles installations du Laboratoire en Sécurité des Réseaux Informatiques (LSRI), financées par la Fondation Canadienne de l'Innovation, qui seront déployées prochainement devraient rendre ce genre d'expériences possibles.

L'évaluation des performances des maliciels est un domaine de recherche récent et en pleine évolution. Des centaines d'interrogations apparaissent et peu d'entre elles ont déjà été répondues. Les recherches sur l'évaluation technique comme la rétro ingénierie et l'observation de trafic réseau sont primordiales mais il sera aussi important de porter attention aux considérations théoriques comme les modèles de propagation épidémique utilisés dans d'autres disciplines. Le partage de connaissances avec d'autres domaines de la science comme la biologie et la sociologie semblent incontournables pour améliorer notre protection contre les maliciels.

Références

BEN-ARTZI, U. et STRORMBERG, D. (2003). A survey of prevention techniques against SQL injection. *Departement of Computer and Systems Sciences, Royal Institute of Technology.*

BOYD, J. (1987). A discourse on winning and losing. Unpublished briefing slides.

BRUNNER, J. (1975). *The Shockwave Rider.* Del Rey, New York.

BUREAU, P.-M. et FERNANDEZ, J. M. (2006). Optimising malware. *Malware '06 : Proceedings of the 2006 IEEE Workshop on Swarm Intelligence.* 34–45.

CASTAÑEDA, F., SEZER, E. C. et XU, J. (2004). Worm vs. worm : preliminary study of an active counter-attack mechanism. *Proceedings of the 2004 ACM Workshop on Rapid malcode (WORM).* ACM Press, New York, NY, USA, 83–93.

CGI SECURITY (2002). Anatomy of the web application worm. http ://www.cgisecurity.com/articles/worms.shtml.

CURRAN, K., MORRISSEY, C., FAGAN, C., MURPHY, C., O'DONNELL, B., FITZPATRICK, G. et CONDIT, S. (2005). Monitoring hacker activity with a honeynet. *International Journal on Network Management*, 15, 123–134.

DENNING, D. E. (1987). An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13, 222–232.

FIREW0RKER (2004). Kernel-mode backdoors for Windows NT. *Phrack Magazine*, 11, 62.

FORCES CANADIENNES (2005). Doctrine interarmée. http ://www.dcds.forces.gc.ca/jointDoc/.

FRAUENTHAL, J. (1980). *Mathematical Modeling in Epidemiology.* Springer-Verlag, New-York, États-Unis.

GARBER, L. (1999). Melissa virus creates a new type of threat. *IEEE Computer*, 32, 16–19.

GUEVARA, E. (1961). *On Guerilla Warfare.* Praeger, Westport, États-Unis.

HAGINO, J.-I. (2006). IPv6 security. CanSecWest 2004.

HANHUA, A. K. (2005). The effect of DNS delays on worm propagation in an ipv6 internet.

HAUSER, V. (2006). Attacking the IPv6 protocol suite. CanSecWest 2006.

HOLZ, T. (2005). A short visit to the bot zoo [malicious bots software]. *IEEE Security and Privacy Magazine*, 3, 76–79.

HOMER (1998). *The Iliad.* Penguin Classics, New-York, États-Unis.

KERMACK, W. O. et MCKENDRICK, A. G. (1927). A Contribution to the Mathematical Theory of Epidemics. *Royal Society of London Proceedings Series A*, 115, 700–721.

KLOG (2000). Backdooring binary objects. *Phrack Magazine*, 10.

LITCHFIELD, D. (2002). OpenRowSet buffer overflows. http ://www.ngssoftware.com/advisories/mssql-ors.txt.

MCCARTY, B. (2003). Botnets : big and bigger. *IEEE Security and Privacy Magazine*, 1, 87–90.

MICROSOFT SECURITY RESPONSE TEAM (2001). Microsoft security bulletin ms01-033. http ://www.microsoft.com/technet/security/bulletin/MS01-033.mspx.

MICROSOFT SECURITY RESPONSE TEAM (2003). Security response team alert - new worm : W32.slammer. http ://www.microsoft.com/technet/security/alerts/slammer.mspx.

MOORE, D., PAXSON, V., SAVAGE, S., SHANNON, C., STANIFORD, S. et WEAVER, N. (2003a). Inside the slammer worm. *IEEE Security and Privacy*, 1, 33–39.

MOORE, D., SHANNON, C. et K CLAFFY (2002). Code-Red : a case study on the spread and victims of an internet worm. *IMW '02 : Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment.* ACM Press, New York, NY, USA, 273–284.

MOORE, D., SHANNON, C., VOELKER, G. M. et SAVAGE, S. (2003b). Network telescopes. Rapport technique, CAIDA.

MOORE, H. D. (2003). Metasploit framework. http ://www.metasploit.com.

NACHENBERG, C. (1997). Computer virus-antivirus coevolution. *Commununiications of the ACM*, 40, 46–51.

NAZARIO, J. (2003a). The blaster worm, the view from 1000 feet away. http ://www.nanog.org/mtg-0310/nazario.html.

NAZARIO, J. (2003b). *Defense and Detection Strategies against Internet Worms.* Artech House, Inc., Norwood, MA, USA.

NAZARIO, J., ANDERSON, J., WALSH, R. et CONNELLY, C. (2001). The future of internet worms.

PANG, R., YEGNESWARAN, V., BARFORD, P., PAXSON, V. et PETERSON, L. (2004). Characteristics of Internet background radiation. *IMC '04 : Proceedings of the 4th ACM SIGCOMM conference on Internet measurement.* ACM Press, New York, NY, USA, 27–40.

PTACEK, T. H. et NEWSHAM, T. N. (1998). Insertion, evasion, and denial of service : Eluding network intrusion detection. Rapport technique, Secure Networks, Inc.

RESCORLA, E. (2003). Security holes...who cares ? *Proceedings of the 11th USENIX Security Symposium.* USENIX, 75–90.

RILEY HASSELL, R. P. (2001). Microsoft internet information services remote buffer overflow. http ://www.eeye.com/html/research/advisories/AD20010618.html.

RIORDAN, J., ZAMBONI, D. et DUPONCHEL, Y. (2005). Lessons learned from Billy Goat, an accurate worm-detection system. Technical Report RZ3609.

SCOTT, D. et SHARP, R. (2002). Developping secure web applications. *IEEE Internet Computing.*

SHARK, S. (1986). Unix trojan horses. *Phrack Magazine*, 1, 7.

SHOCH, J. F. et HUPP, J. A. (1982). The worm programs early experience with a distributed computation. *Commun. ACM*, 25, 172–180.

SKOUDIS, E. et ZELTSER, L. (2003). *Malware : Fighting Malicious Code.* Prentice Hall PTR.

SNORT (2002). Snort, the open source network intrusion detection system. http ://www.snort.org.

SPAFFORD, E. H. (1989). The Internet worm program : An analysis. *Computer Communication Review*, 19.

SPI LABORATORIES (2002). SQL injection, are your web applications secure ? http ://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf.

STAFFORD, T. et URBACZEWSKI, A. (2004). Spyware : The ghost in the machine. *Communications of the Association for Information Systems.*

STANIFORD, S., PAXSON, V. et WEAVER, N. (2002). How to own the Internet in your spare time. *Proceedings of the 11th USENIX Security Symposium.* USENIX Association, Berkeley, CA, USA, 149–167.

THE HONEYNET PROJECT (2002). *Know your Enemy.* Addison-Wesley, Boston, États-Unis.

TODD, M. (2003). Worms as attack vectors : Theory, threats, and defenses.

WAGNER, A., UBENDORFER, T., PLATTNER, B. et HIESTAND, R. (2003). Experiences with worm propagation simulations.

WEAVER, N., PAXSON, V. et STANIFORD, S. (2003a). A worst-case worm, technical report. http ://www.silicondefense.com/research/worms/worstcase.pdf.

WEAVER, N., PAXSON, V., STANIFORD, S. et CUNNINGHAM, R. (2003b). A taxonomy of computer worms. *Proceedings of the 2003 ACM workshop on Rapid Malcode (WORM).* ACM Press, New York, NY, USA, 11–18.

WILEY, B. (2002). Curious yellow : The first coordinated worm design. http ://-blanu.net/curious_yellow.html.

ZALEWSKI, M. (2000). Writing internet worms for fun and profit.

ZALEWSKI, M. (2001). Against the system : Rise of the robots. *Phrack Magazine*, 5.

ZOU, C. C., GONG, W. et TOWSLEY, D. (2002). Code red worm propagation modeling and analysis. *CCS '02 : Proceedings of the 9th ACM conference on Computer and communications security*. ACM Press, New York, NY, USA, 138–147.

ZOU, C. C., TOWSLEY, D., GONG, W. et CAI, S. (2005). Routing worm : A fast, selective attack worm based on ip address information. *PADS '05 : Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*. IEEE Computer Society, Washington, DC, USA, 199–206.

ZOVI, D. (2001). ADMutate. http ://www.ktwo.ca/.